Developer Guide

AWS Elastic Beanstalk

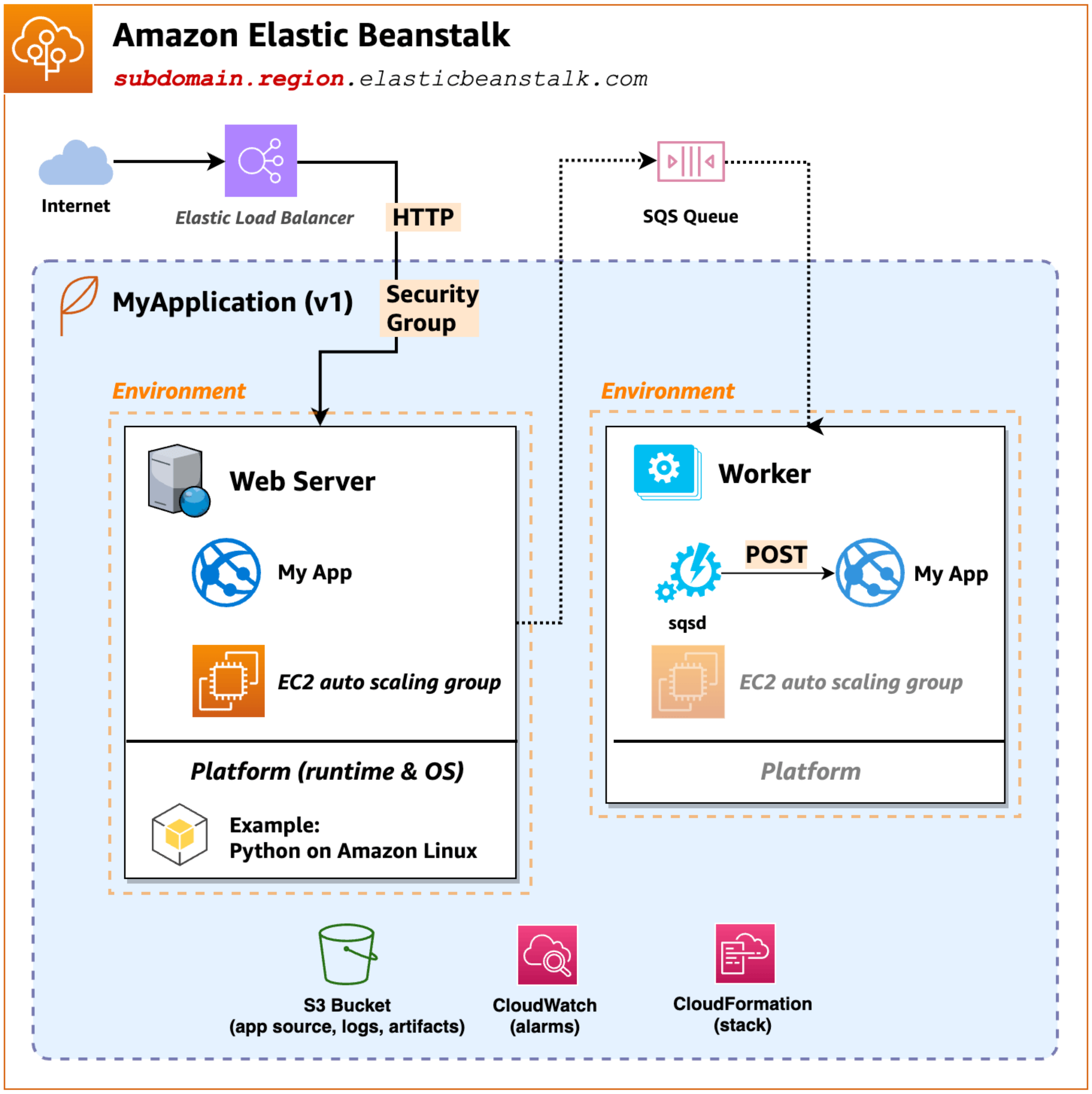
AWS Elastic Beanstalk: Developer Guide

Table of Contents

What is AWS Elastic Beanstalk?

With Elastic Beanstalk you can deploy web applications into the AWS Cloud on a variety of supported platforms. You build and deploy your applications. Elastic Beanstalk provisions Amazon EC2 instances, configures load balancing, sets up health monitoring, and dynamically scales your environment.

In addition to *web server* environments, Elastic Beanstalk also provides *worker* environments which you can use to process messages from an Amazon SQS queue, useful for asynchronous or long-running tasks. For more information, see Elastic Beanstalk worker environments.



Supported platforms

Elastic Beanstalk supports applications developed in Go, Java, .NET, Node.js, PHP, Python, and Ruby. Elastic Beanstalk also supports Docker containers, where you can choose your own programming language and application dependencies. When you deploy your application, Elastic

Beanstalk builds the selected supported platform version and provisions one or more AWS resources, such as Amazon EC2 instances, in your AWS account to run your application.

You can interact with Elastic Beanstalk through the Elastic Beanstalk console, the AWS Command Line Interface (AWS CLI), or the EB CLI, a high-level command line tool designed specifically for Elastic Beanstalk.

You can perform most deployment tasks, such as changing the size of your fleet of Amazon EC2 instances or monitoring your application, directly from the Elastic Beanstalk web interface (console).

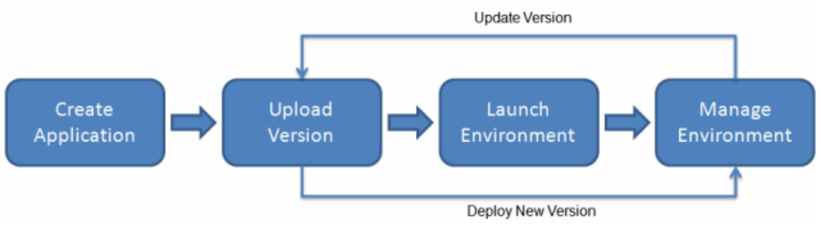
To learn more about how to deploy a sample web application using Elastic Beanstalk, see Learn how to get started with Elastic Beanstalk.

Application deploy workflow

To use Elastic Beanstalk, you create an application, then upload your application source bundle to Elastic Beanstalk. Next, you provide information about the application, and Elastic Beanstalk automatically launches an environment and creates and configures the AWS resources needed to run your code.

After you create and deploy your application and your environment is launched, you can manage your environment and deploy new application versions. Information about the application—including metrics, events, and environment status—is made available through the Elastic Beanstalk console, APIs, and Command Line Interfaces.

The following diagram illustrates Elastic Beanstalk workflow:



Pricing

There is no additional charge for Elastic Beanstalk. You pay only for the underlying AWS resources that your application consumes. For details about pricing, see the Elastic Beanstalk service detail page.

Next steps

We recommend the tutorial, Getting started tutorial, to start using Elastic Beanstalk. The tutorial steps you through creating, viewing, and updating a sample Elastic Beanstalk application.

Learn how to get started with Elastic Beanstalk

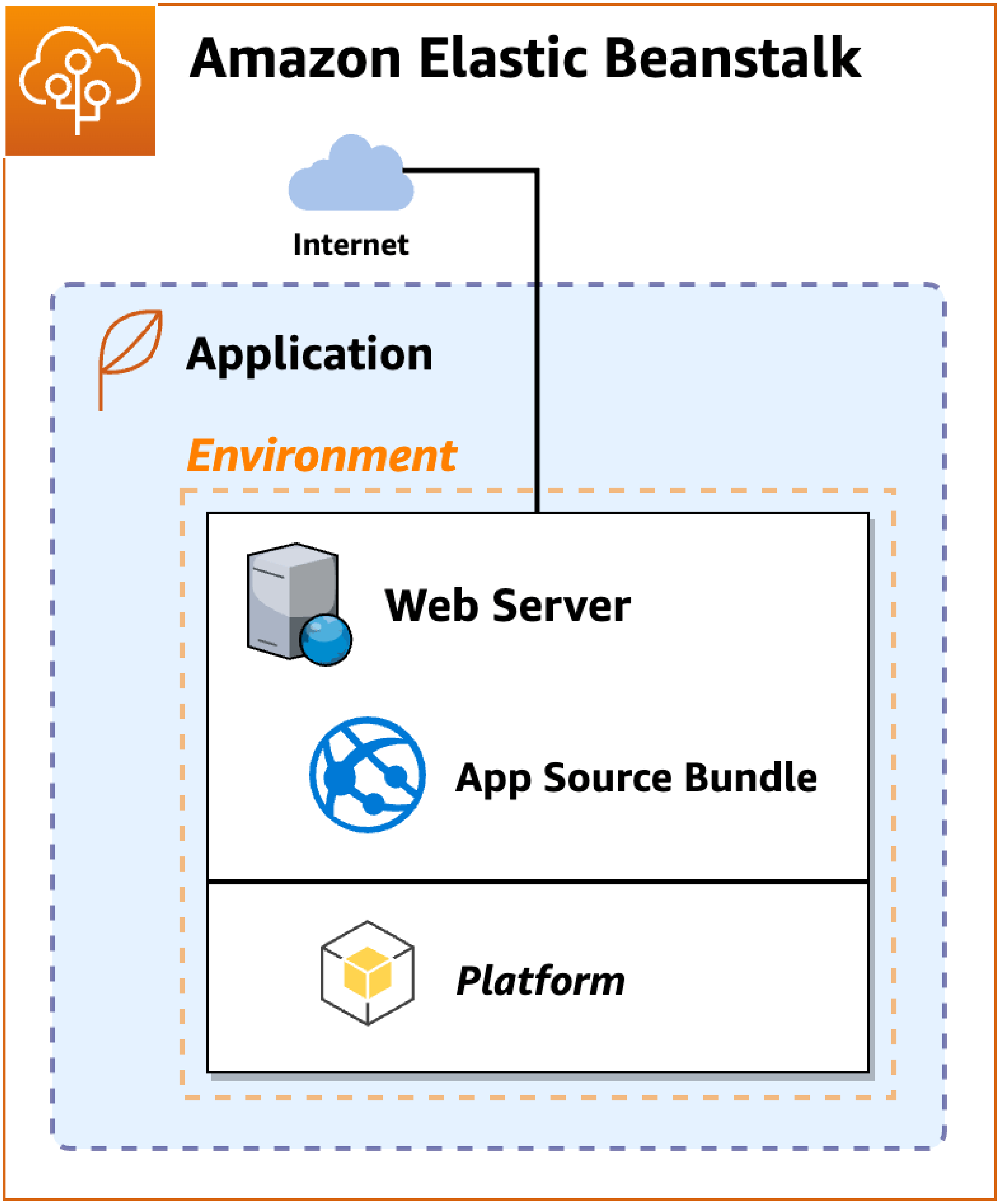
With Elastic Beanstalk you can deploy, monitor, and scale web applications and services. Typically, you will develop your code locally then deploy it to Amazon EC2 server instances. Theses instances, also called *environments*, run on *platforms* that can be upgraded through the AWS console or the command line.

To get started, we recommend deploying a pre-built sample application directly from the console. Then, you can learn how to develop locally and deploy from the command line in the the section called “QuickStart for PHP”.

There is no cost for using Elastic Beanstalk, but standard fees do apply to AWS resources that you create during the course of this tutorial until you delete them at the end. The total charges are typically less than a dollar. For information about how to minimize charges, see AWS free tier.

After completing this tutorial, you will understand the basics of creating, configuring, deploying, updating, and monitoring an Elastic Beanstalk application with environments running on Amazon EC2 instances.

Estimated duration: **35-45 minutes**


Amazon Elastic Beanstalk
Internet
Application
Environment
Web Server
App Source Bundle
Platform

What you will build

Your first Elastic Beanstalk application will consist of a single Amazon EC2 environment running the PHP sample on a PHP managed platform.

Elastic Beanstalk application

An *Elastic Beanstalk application* is a container for Elastic Beanstalk components, including *environments* where your application code runs on *platforms* provided and managed by Elastic Beanstalk, or in custom containers that you provide.

Environment

An Elastic Beanstalk *environment* is a collection of AWS resources running together including an Amazon EC2 instance. When you create an environment, Elastic Beanstalk provisions the necessary resources into your AWS account.

Platform

A *platform* is a combination of an operating system, programming language runtime, web server, application server, and additional Elastic Beanstalk components. Elastic Beanstalk provides manged platforms, or you can provide your own platform in a container.

Elastic Beanstalk supports platforms for different programming languages, application servers, and Docker containers. When you create an environment, you must choose the platform. You can upgrade the platform, but you cannot **change** the platform for an environment.

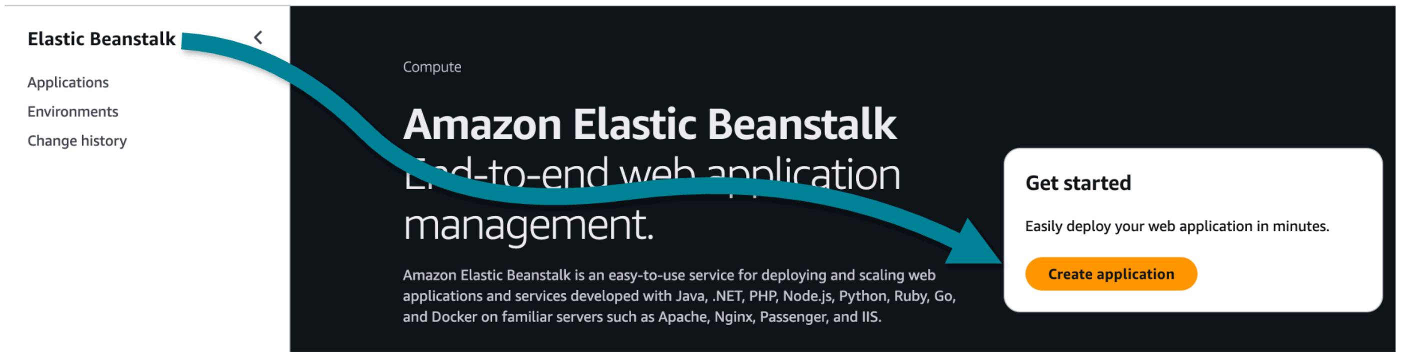
Switching platforms

If you need to change programming languages, you must create and switch to a new environment on a different platform.

Step 1 - Create an application

To create your example application, you'll use the **Create application** console wizard. It creates an Elastic Beanstalk application and launches an environment within it.

Reminder: an *environment* is a collection of AWS resources required to run your application code.



Amazon Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications and services developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS.

To create an application

1. Open the Elastic Beanstalk console.
2. Choose **Create application**.
3. For **Application name** enter **`getting-started-app`**.

The console provides a six step process for creating an application and configuring an environment. For this quick start, you'll only need to focus on the first two steps, then you can skip ahead to review and create your application and environment.

To configure an environment

1. In **Environment information**, for **Environment name** enter: **`gs-app-web-env`**.
2. For **Platform**, choose the **PHP** platform.
3. For **Application code** and **Presets**, accept the defaults (*Sample application* and *Single instance*), then choose **Next**.

To configure service access

Next, you need two roles. A *service role* allows Elastic Beanstalk to monitor your EC2 instances and upgrade you environment's platform. An *EC2 instance profile* role permits tasks such as writing logs and interacting with other services.

To create the Service role

1. For **Service role**, choose **Create role**.
2. For **Trusted entity type**, choose **AWS service**.

3. For **Use case**, choose **Elastic Beanstalk – Environment**.
4. Choose **Next**.
5. Verify that **Permissions policies** include the following, then choose **Next**:
   - `AWSElasticBeanstalkEnhancedHealth`
   - `AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy`
6. Choose **Create role**.
7. Return to the **Configure service access** tab, refresh the list, then select the newly created service role.

To create the EC2 instance profile

1. Choose **Create role**.
2. For **Trusted entity type**, choose **AWS service**.
3. For **Use case**, choose **Elastic Beanstalk – Compute**.
4. Choose **Next**.
5. Verify that **Permissions policies** include the following, then choose **Next**:
   - `AWSElasticBeanstalkWebTier`
   - `AWSElasticBeanstalkWorkerTier`
   - `AWSElasticBeanstalkMulticontainerDocker`
6. Choose **Create role**.
7. Return to the **Configure service access** tab, refresh the list, then select the newly created EC2 instance profile.

To finish configuring and creating your application

1. Skip over **EC2 key pair**.

   We'll show you other ways to connect to your Amazon EC2 instances through the Console.
2. Choose **Skip to Review** to move over several optional steps.

   Optional steps: networking, databases, scaling parameters, advanced configuration for updates, monitoring, and logging.
3. On the **Review** page which shows a summary of your choices, choose **Submit**.

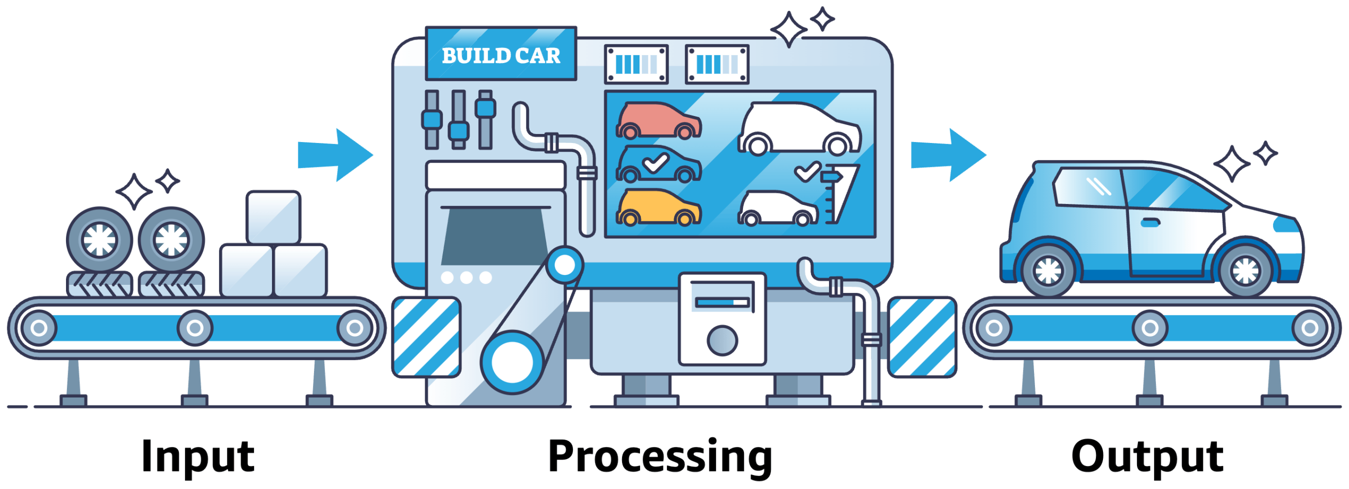
Congratulations!

You have created an application and configured an environment! Now you need to wait for the resources to deploy.

Step 2 - Deploy your application

When you create an application, Elastic Beanstalk sets up the environments for you. You just need to sit back and wait.

The initial deploy can take up to five minutes to create the resources. Updates will take less time because only changes will be deployed to your stack.



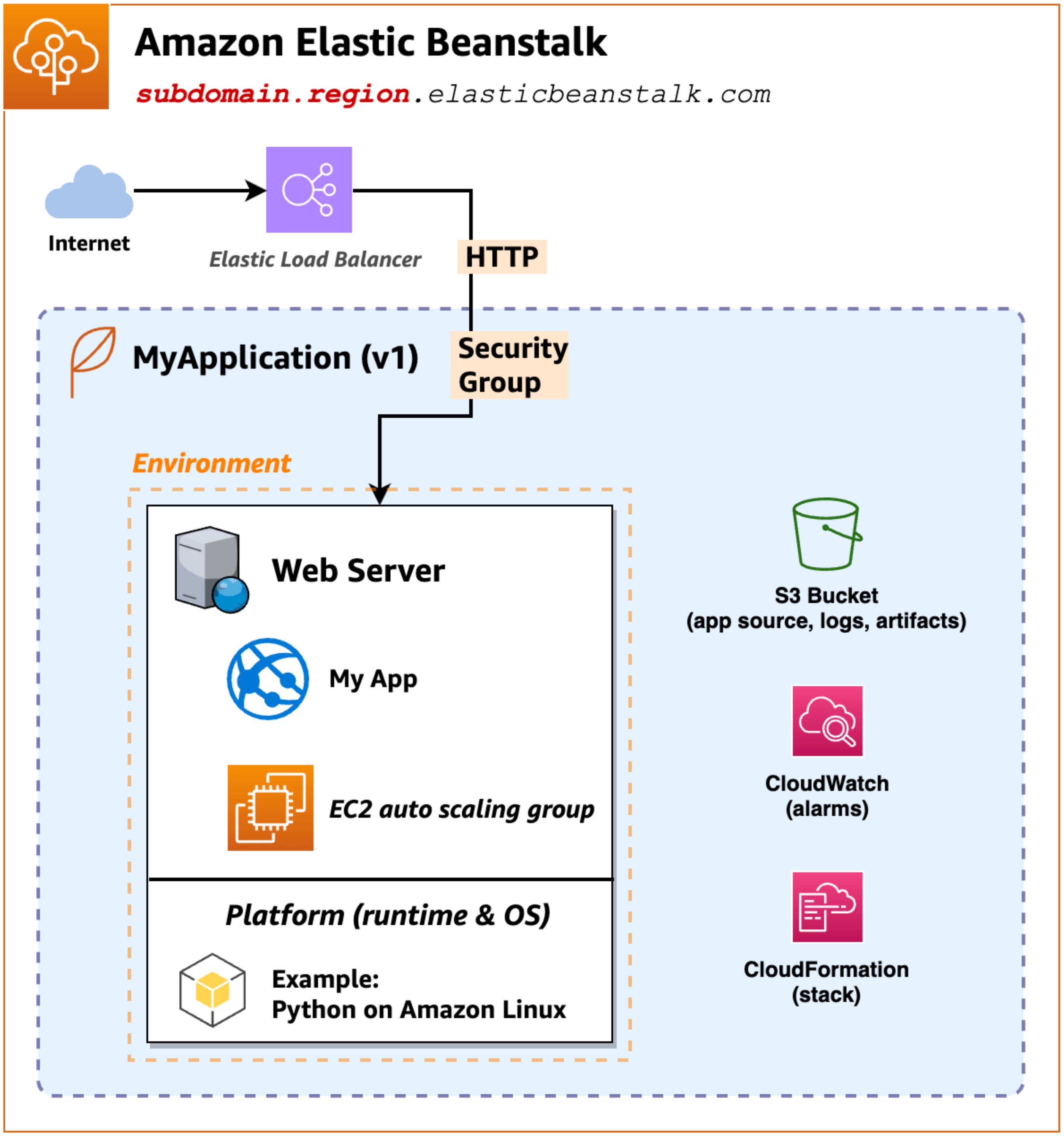
When you create the example application, Elastic Beanstalk creates the following resources:

- **EC2 instance –** An Amazon EC2 virtual machine configured to run web apps on the platform you selected.

  Every platform runs a different set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination thereof. Most platforms use either Apache or nginx as a reverse proxy to forward web traffic to your web app, serve static assets, and generate access and error logs. You can connect to your Amazon EC2 instances to view configuration and logs.

- **Instance security group –** An Amazon EC2 security group will be created to allow incoming requests on port 80, so inbound traffic on a load balancer can reach your web app.
- **Amazon S3 bucket –** A storage location for your source code, logs, and other artifacts.
- **Amazon CloudWatch alarms –** Two CloudWatch alarms are created to monitor the load on your instances and scale them up or down as needed.
- **AWS CloudFormation stack –** Elastic Beanstalk uses AWS CloudFormation to deploy the resources in your environment and make configuration changes. You can view the resource definition template in the AWS CloudFormation console.
- **Domain name –** A domain name that routes to your web app in the form : `subdomain.region.elasticbeanstalk.com`.

Elastic Beanstalk creates your application, launches an environment, makes an application version, then deploys your code into the environment. During the process, the console tracks progress and displays event status in the Events tab.



After all of the resources are deployed, the environment's health should change to **Ok**.

Your application is ready!

After you see your application health change to **Ok**, you can browse to your web application's website.

Step 3 - Explore the Elastic Beanstalk environment

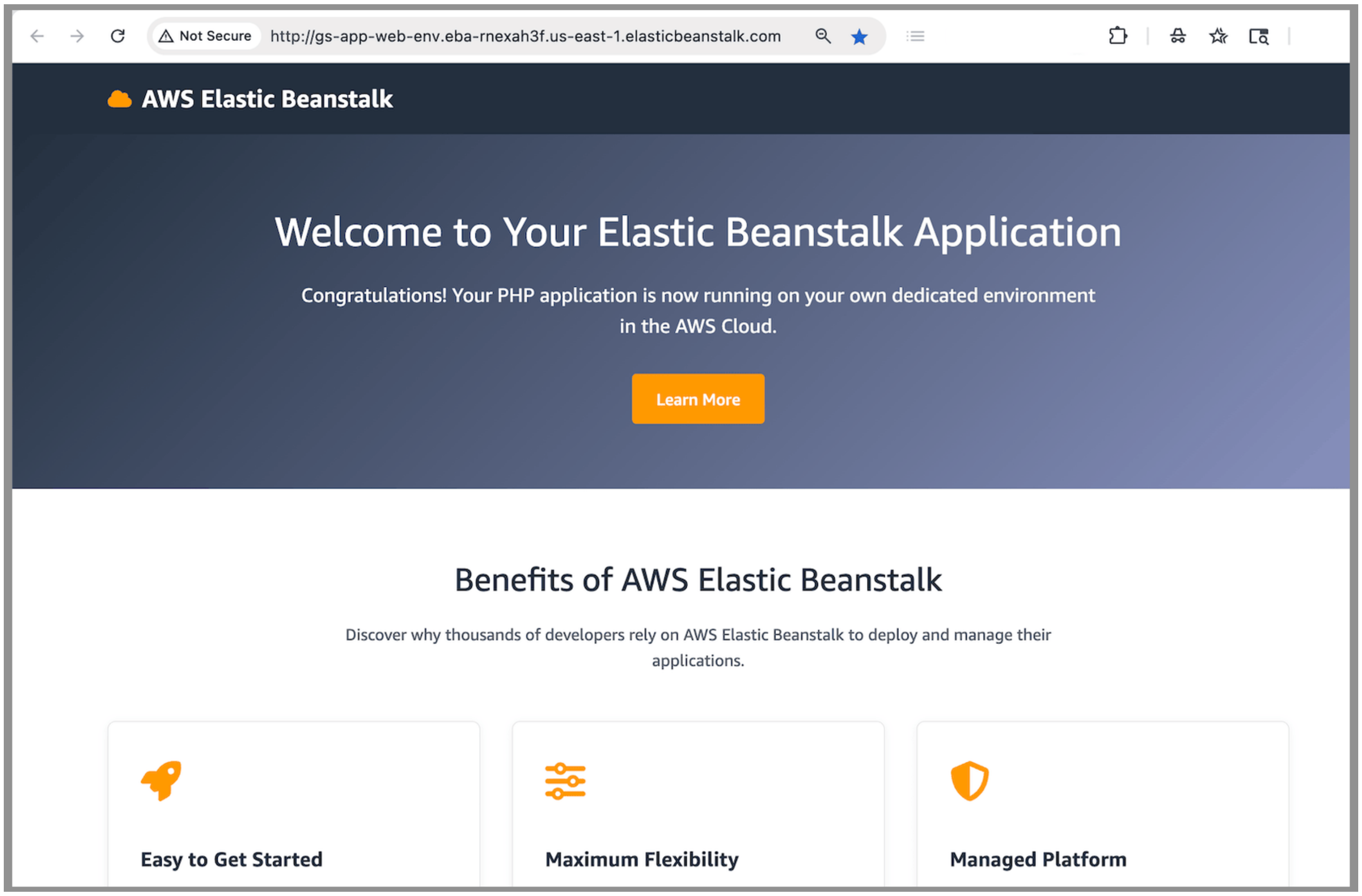
You'll start exploring your deployed application environment from the **Environment overview** page in the console.

To view the environment and your application

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Go to environment** to browse your application!

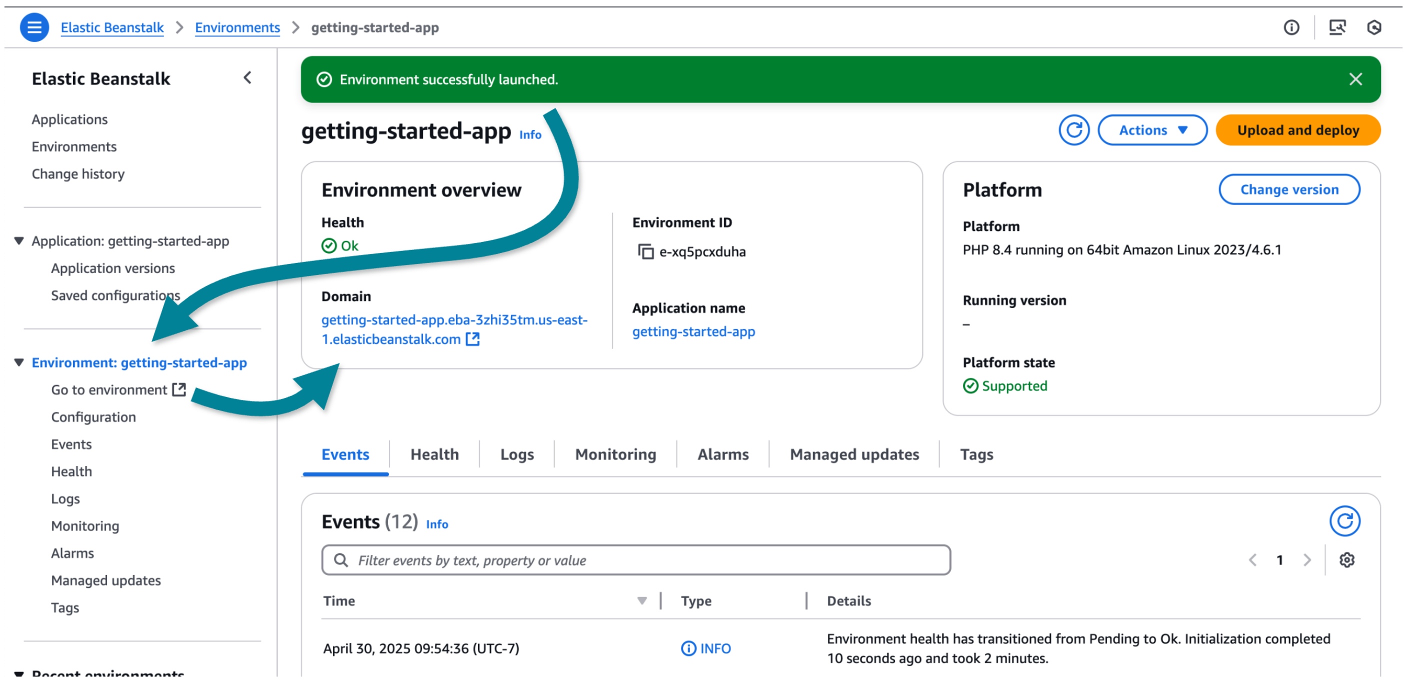
   (You can also choose the URL link listed for **Domain** to browse your application.)

   The connection will be HTTP (not HTTPS), so you might see a warning in your browser.


Not Secure
http://gs-app-web-env.eba-rnexah3f.us-east-1.elasticbeanstalk.com
AWS Elastic Beanstalk
Welcome to Your Elastic Beanstalk Application
Congratulations! Your PHP application is now running on your own dedicated environment in the AWS Cloud.
Learn More
Benefits of AWS Elastic Beanstalk
Discover why thousands of developers rely on AWS Elastic Beanstalk to deploy and manage their applications.
Easy to Get Started
Maximum Flexibility
Managed Platform

Back in the Elastic Beanstalk console, the upper portion shows the **Environment overview** with top level information about your environment, including name, domain URL, current health status, running version, and the platform that the application is running on. The running version and platform are essential for troubleshooting your currently deployed application.

After the overview pane, you will see recent environment activity in the **Events** tab.



Elastic Beanstalk > Environments > getting-started-app

Elastic Beanstalk

Applications
Environments
Change history

Application: getting-started-app
Application versions
Saved configurations

Environment: getting-started-app
Go to environment
Configuration
Events
Health
Logs
Monitoring
Alarms
Managed updates
Tags

Recent environments

Environment successfully launched.

getting-started-app Info

Actions | Upload and deploy

Environment overview

Health
Ok

Environment ID
e-xq5pcxduha

Domain
getting-started-app.eba-3zhi35tm.us-east-1.elasticbeanstalk.com

Application name
getting-started-app

Platform | Change version

Platform
PHP 8.4 running on 64bit Amazon Linux 2023/4.6.1

Running version
–

Platform state
Supported

Events | Health | Logs | Monitoring | Alarms | Managed updates | Tags

Events (12) Info

Filter events by text, property or value

Time	Type	Details
April 30, 2025 09:54:36 (UTC-7)	INFO	Environment health has transitioned from Pending to Ok. Initialization completed 10 seconds ago and took 2 minutes.

While Elastic Beanstalk creates your AWS resources and launches your application, the environment is in a `Pending` state. Status messages about launch events are continuously added to the list of **Events** .

The environment's **Domain** is the URL for your deployed web application. In the left navigation pane, **Go to environment** also takes you to your domain. Similarly, the left navigation pane has links that correspond to the various tabs.

Take note of the **Configuration** link in the left navigation pane. which displays a summary of environment configuration option values, grouped by category.

Environment configuration settings

Take note of the **Configuration** link in the left navigation pane. You can view and edit detailed environment settings, such as service roles, networking, database, scaling, managed platform updates, memory, health monitoring, rolling deployment, logging, and more!

The various tabs contain detailed information about your environment:

- **Events –** View an updating list of information and error messages from the Elastic Beanstalk service and other services for resources in your environment.
- **Health –** View status and detailed health information for the Amazon EC2 instances running your application.
- **Logs –** Retrieve and download logs from the Amazon EC2 in your environment. You can retrieve full logs or recent activity. The retrieved logs are available for 15 minutes.
- **Monitoring –** View statistics for the environment, such as average latency and CPU utilization.
- **Alarms –** View and edit alarms that are configured for environment metrics.
- **Managed updates –** View information about upcoming and completed managed platform updates and instance replacement.
- **Tags –** View and edit key-value pairs that are applied to your environment.

Note

Links in the console navigation pane will display the corresponding tab.

Troubleshooting with logs

For troubleshooting unexpected behaviors or debugging deployments, you might want to check the logs in your environments.

You can request 100 lines of all the log files under the **Logs** tab in the Elastic Beanstalk console. Alternatively, you can connect directly to the Amazon EC2 instance and tail the logs in realtime.

To request the logs (Elastic Beanstalk console)

1. Navigate to your environment in the Elastic Beanstalk console.
2. Choose the **Logs** tab or left-nav, then choose **Request logs**.
3. Select **Last 100 lines**.
4. After the logs are created, choose the **Download** link to view the logs in the browser.

In the logs, find the log and note the directory for the nginx access log.

Add a policy to enable connections to Amazon EC2

Before you can connect, you must add a policy that enables connections to Amazon EC2 with Session Manager.

1. Navigate to the IAM console.
2. Find and select the **`aws-elasticbeanstalk-ec2-role`** role.
3. Choose **Add permission**, then **Attach policies**.
4. Search for a *default policy* that **begins** with the following text: **`AmazonSSMManagedEC2Instance`**, then add it to the role.

To connect to your Amazon EC2 with Session Manager

1. Navigate to the Amazon EC2 console.
2. Choose **Instances**, then select your `gs-app-web-env` instance.
3. Choose **Connect**, then **Session Manager**.
4. Choose **Connect**.

After connecting to the instance, start a bash shell and tail the logs:

1. Run the command **`bash`**.
2. Run the command **`cd /var/log/nginx`**.
3. Run the command **`tail -f access.log`**.
4. In your browser, go to the application domain URL. Refresh.

Congratulations, you're connected!

You should see log entries in your instance update every time you refresh the page.

Connect button not working?

If the connect button is not available, go back to IAM and verify that you added the necessary policy to the role.

Step 4 - Update your application

Eventually, you will want to update your application. You can deploy a new version at any time, as long as no other update operations are in progress on your environment.

The application version that you started this tutorial with is called **Sample Application**.

To update your application version

1. Download the following PHP sample application:

   PHP – php-v2.zip
2. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
3. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
4. On the environment overview page, choose **Upload and deploy**.
5. Select **Choose file**, and then upload the sample application source bundle that you downloaded.

   The console automatically fills in the **Version label** with a new unique label, automatically incrementing a trailing integer. If you choose your own version label, ensure that it's unique.

6. Choose **Deploy**.

While Elastic Beanstalk deploys your file to your Amazon EC2 instances, you can view the deployment status on the **Environment overview** page. While the application version is updated, the environment **Health** status is gray. When the deployment is complete, Elastic Beanstalk performs an application health check. When the application responds to the health check, it's considered healthy and the status returns to green. The environment overview shows the new **Running Version**—the name you provided as the **Version label**.

Elastic Beanstalk also uploads your new application version and adds it to the table of application versions. To view the table, choose **Application versions** under **getting-started-app** on the navigation pane.

Update success!

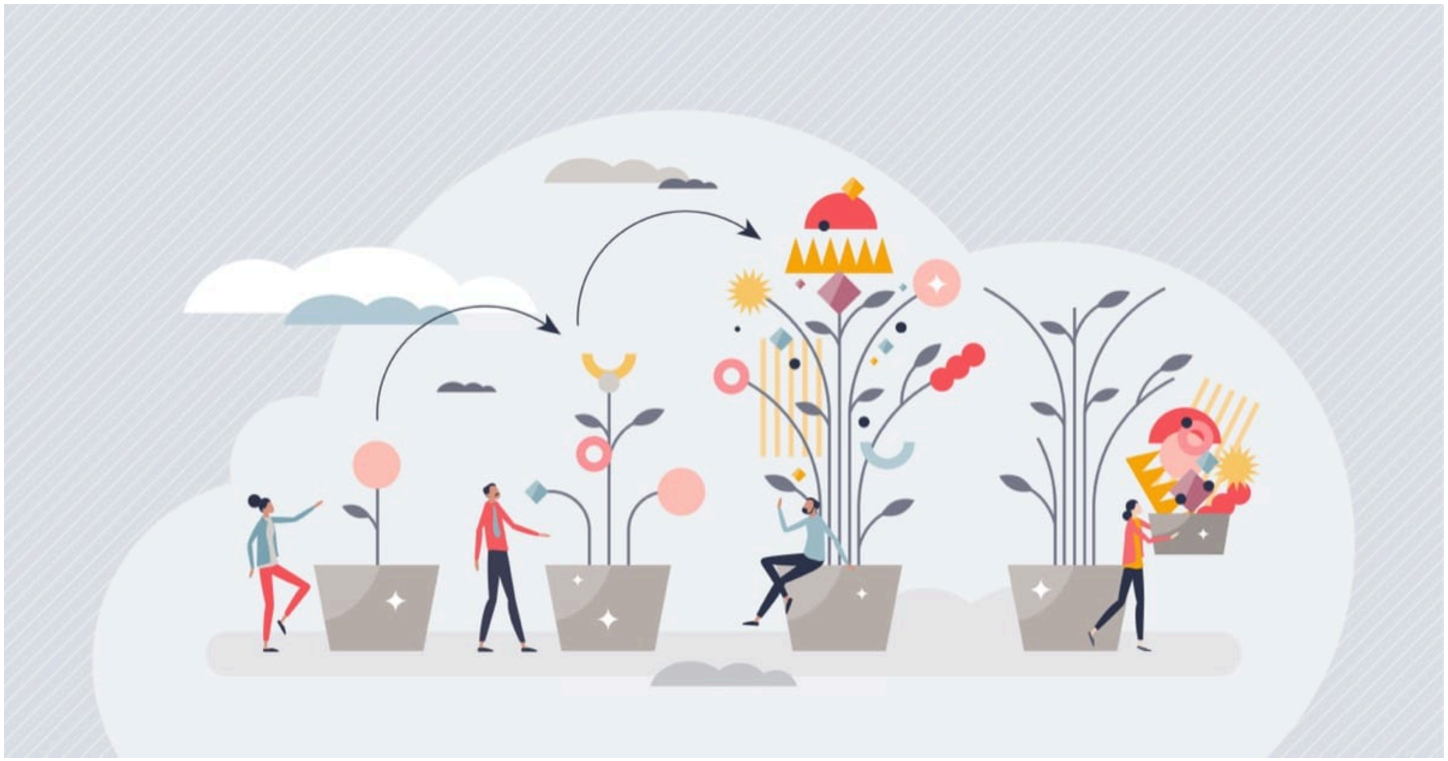
You should see an updated "v2" message after refreshing your browser.
If you want to edit the source yourself, unzip, edit, then re-zip the source bundle. On macOS, use the following command from inside your php directory with the -X to exclude extra file attributes:

```
zip -X -r ../php-v2.zip .
```

Step 5 - Scale your application

You can configure your environment to better suit your application. For example, if you have a compute-intensive application, you can change the type of Amazon Elastic Compute Cloud (Amazon EC2) instance that is running your application. To apply configuration changes, Elastic Beanstalk performs an environment update.

Some configuration changes are simple and happen quickly. Some changes require deleting and recreating AWS resources, which can take several minutes. When you change configuration settings, Elastic Beanstalk warns you about potential application downtime.



Increase capacity settings

In this example of a configuration change, you edit your environment's capacity settings. You configure a load-balanced, scalable environment that has between two and four Amazon EC2 instances in its Auto Scaling group, and then you verify that the change occurred. Elastic Beanstalk creates an additional Amazon EC2 instance, adding to the single instance that it created initially. Then, Elastic Beanstalk associates both instances with the environment's load balancer. As a result, your application's responsiveness is improved and its availability is increased.

To change your environment's capacity

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Instance traffic and scaling** configuration category, choose **Edit**.
5. Collapse the **Instances** section, so you can more easily see the **Capacity** section. Under **Auto Scaling group** change **Environment type** to **Load balanced**.
6. In the **Instances** row, change **Min** to **2** and **Max** to **4**.

7. To save the changes choose **Apply** at the bottom of the page.

   *If you are warned that the update will replace all of your current instances. Choose **Confirm**.*

The environment update can take a few minutes. You should see several updates in the list of events. Watch for the event **Successfully deployed new configuration to environment**.

Verify increased capacity

After the environment update is complete and the environment is ready, Elastic Beanstalk automatically launched a second instance to meet your new minimum capacity setting.

To verify the increased capacity

1. Choose **Health** from either the tab list or left navigation pane.
2. Review the **Enhanced instance health** section.

You just scaled up!

With two Amazon EC2 instances, your environment capacity has doubled, and it only took a few minutes.

Cleaning up your Elastic Beanstalk environment

To ensure that you're not charged for any services you aren't using, delete all application versions and terminate environments, which also deletes the AWS resources that the environment created for you.

To delete the application and all associated resources

1. Delete all application versions.

   a. Open the [Elastic Beanstalk console](), and in the **Regions** list, select your AWS Region.

   b. In the navigation pane, choose **Applications**, and then choose **getting-started-app**.

   c. In the navigation pane, find your application's name and choose **Application versions**.

   d. On the **Application versions** page, select all application versions that you want to delete.

   e. Choose **Actions**, and then choose **Delete**.

   f. Turn on **Delete versions from Amazon S3**.

   g. Choose **Delete**, and then choose **Done**.

2. Terminate the environment.

   a. In the navigation pane, choose **getting-started-app**, and then choose **GettingStartedApp-env** in the environment list.

   b. Choose **Actions**, and then choose **Terminate Environment**.
   c. Confirm that you want to terminate **GettingStartedApp-env** by typing the environment name, and then choose **Terminate**.

3. Delete the getting-started-app application.

   a. In the navigation pane, choose the **getting-started-app**.
   b. Choose **Actions**, and then choose **Delete application**.
   c. Confirm that you want to delete **getting-started-app** by typing the application name, and then choose **Delete**.

Congratulations!

You have successfully deployed a sample application to the AWS Cloud, uploaded a new version, modified its configuration to add a second Auto Scaling instance, and cleaned up your AWS resources!

Next steps

To learn how to use the **eb** command line tool to automate deploying your code to Elastic Beanstalk, We suggest continuing with the the section called "QuickStart for PHP".

Next, you might want to review how to set up HTTPS connection, see the section called "HTTPS".

Setting up the EB command line interface (EB CLI) to manage Elastic Beanstalk

The EB CLI is a command line interface which provides interactive commands to create, update, and monitor environments in AWS Elastic Beanstalk. The EB CLI open-source project is on Github: `aws/aws-elastic-beanstalk-cli`

After you install the EB CLI and configure a project directory, you can create environments with a single command:

```
$ eb create <my-beanstalk-environment>
```

We recommend installing with the setup script, learn how in the section called “Install EB CLI”.

The AWS CLI provides direct access to low-level Elastic Beanstalk APIs. Although powerful, it is also verbose and less preferred over the EB CLI. For example, creating an environment with the AWS CLI requires the following series of commands:

```
$ aws elasticbeanstalk check-dns-availability \
    --cname-prefix my-cname
$ aws elasticbeanstalk create-application-version \
    --application-name my-application \
    --version-label v1 \
    --source-bundle S3Bucket=amzn-s3-demo-bucket,S3Key=php-proxy-sample.zip
$ aws elasticbeanstalk create-environment \
    --cname-prefix my-cname \
    --application-name my-app \
    --version-label v1 \
    --environment-name my-env \
    --solution-stack-name "64bit Amazon Linux 2023 v4.5.0 running Ruby 3.4"
```

Install EB CLI with setup script (recommended)

We recommend the installer script

We recommend using the installer script to set up the EB CLI and its dependencies and prevent potential conflicts with other Python packages.

Pre-requisites: Git, Python, and virtualenv

To download and use the installer script

1. Clone the repository.

   ```
   git clone https://github.com/aws/aws-elastic-beanstalk-cli-setup.git
   ```

2. Install or upgrade the EB CLI.

   macOS / Linux in Bash or Zsh

   ```
   python ./aws-elastic-beanstalk-cli-setup/scripts/ebcli_installer.py
   ```

   Windows in PowerShell or Command window

   ```
   python .\aws-elastic-beanstalk-cli-setup\scripts\ebcli_installer.py
   ```

3. Verify that the EB CLI is installed correctly.

   ```
   $ eb --version
   EB CLI 3.21.0 (Python 3.12)
   ```

For complete installation instructions, see the `aws/aws-elastic-beanstalk-cli-setup` repository on GitHub.

Manually install the EB CLI

You can install the EB CLI on Linux, macOS, and Windows with the `pip` package manager for Python which provides installation, upgrade, and removal of Python packages and their dependencies.

We recommend the installer script

We recommend using the Install EB CLI to set up the EB CLI and prevent dependency conflicts.

Prerequisite - You must have a supported version of Python installed. You can download it from the Python downloads page on the Python website.

To install the EB CLI (manually)

1. Run the following command.

```
$ pip install awsebcli --upgrade --user
```

   The `--upgrade` option tells `pip` to upgrade any requirements that are already installed. The `--user` option tells `pip` to install the program to a subdirectory of your user directory to avoid modifying libraries that your operating system uses.

   Troubleshooting issues

   If you encounter issues when you try to install the EB CLI with `pip`, you can install the EB CLI in a virtual environment to isolate the tool and its dependencies, or use a different version of Python than you normally do.

2. Add the path to the executable file to your PATH variable:

   - On Linux and macOS:

     Linux – `~/.local/bin`

     macOS – `~/Library/Python/`*`3.12`*`/bin`

     To modify your PATH variable (Linux, Unix, or macOS):

     a. Find your shell's profile script in your user folder. If you are not sure which shell you have, run `echo $SHELL`.

```
$ ls -a ~
.  ..  .bash_logout  .bash_profile  .bashrc  Desktop  Documents  Downloads
```

        - **Bash** – `.bash_profile`, `.profile`, or `.bash_login`.
        - **Zsh** – `.zshrc`
        - **Tcsh** – `.tcshrc`, `.cshrc` or `.login`.

   b. Add an export command to your profile script. The following example adds the path represented by *LOCAL_PATH* to the current PATH variable.

      ```
      export PATH=LOCAL_PATH:$PATH
      ```

   c. Load the profile script described in the first step into your current session. The following example loads the profile script represented by *PROFILE_SCRIPT*.

      ```
      $ source ~/PROFILE_SCRIPT
      ```

- On Windows:

  Python 3.12 – `%USERPROFILE%\AppData\Roaming\Python\Python312\Scripts`

  Python earlier versions – `%USERPROFILE%\AppData\Roaming\Python\Scripts`

  To modify your PATH variable (Windows):

  a. Press the Windows key, and then enter **`environment variables`**.
  b. Choose **Edit environment variables for your account**.
  c. Choose **PATH**, and then choose **Edit**.
  d. Add paths to the **Variable value** field, separated by semicolons. For example: *`C:\item1\path`***`;C:\item2\path`**
  e. Choose **OK** twice to apply the new settings.
  f. Close any running Command Prompt windows, and then reopen a Command Prompt window.

3. Verify that the EB CLI installed correctly by running **eb --version**.

   ```
   $ eb --version
   EB CLI 3.21.0 (Python 3.12)
   ```

The EB CLI is updated regularly to add functionality that supports the latest Elastic Beanstalk features. To update to the latest version of the EB CLI, run the installation command again.

```
$ pip install awsebcli --upgrade --user
```

If you need to uninstall the EB CLI, use `pip uninstall`.

```
$ pip uninstall awsebcli
```

Install the EB CLI in a virtual environment

You can avoid version requirement conflicts with other `pip` packages by installing the EB CLI in a virtual environment.

To install the EB CLI in a virtual environment

1. First, install `virtualenv` with `pip`.

   ```
   $ pip install --user virtualenv
   ```

2. Create a virtual environment.

   ```
   $ virtualenv ~/eb-ve
   ```

   To use a Python executable other than the default, use the `-p` option.

   ```
   $ virtualenv -p /usr/bin/python3.12 ~/eb-ve
   ```

3. Activate the virtual environment.

   Linux, Unix, or macOS

   ```
   $ source ~/eb-ve/bin/activate
   ```

   Windows

   ```
   $ %USERPROFILE%\eb-ve\Scripts\activate
   ```

4. Install the EB CLI.

   ```
   (eb-ve)$ pip install awsebcli --upgrade
   ```

5. Verify that the EB CLI is installed correctly.

   ```
   $ eb --version
   EB CLI 3.23.0 (Python 3.12)
   ```

You can use the `deactivate` command to exit the virtual environment. Whenever you start a new session, run the activation command again.

To upgrade to the latest version, run the installation command again.

```
(eb-ve)$ pip install awsebcli --upgrade
```

Install the EB CLI with homebrew

The latest version of the EB CLI is typically available from `Homebrew` a couple of days after it appears in `pip`.

We recommend the installer script

We recommend using the Install EB CLI to set up the EB CLI and prevent dependency conflicts.

To install the EB CLI with Homebrew

1. Ensure you have the latest version of `Homebrew`.

   ```
   $ brew update
   ```

2. Run `brew install awsebcli`.

   ```
   $ brew install awsebcli
   ```

3. Verify that the EB CLI is installed correctly.

   ```
   $ eb --version
   EB CLI 3.21.0 (Python 3.12)
   ```

Configure the EB CLI

After installing the EB CLI, you are ready to configure your project directory and the EB CLI by running **eb init**.The following example shows the configuration steps when running **eb init** for the first time in a project folder named `eb`.

To initialize an EB CLI project

1. First, the EB CLI prompts you to select a region. Choose your preferred region.

```
~/eb $ eb init
Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
3) us-west-2 : US West (Oregon)
4) eu-west-1 : Europe (Ireland)
5) eu-central-1 : Europe (Frankfurt)
6) ap-south-1 : Asia Pacific (Mumbai)
7) ap-southeast-1 : Asia Pacific (Singapore)
...
(default is 3): 3
```

2. If prompted, provide your access key and secret key so that the EB CLI can manage resources for you. Access keys are created in the AWS Identity and Access Management console. If you don't have keys, see How Do I Get Security Credentials? in the *Amazon Web Services General Reference*.

```
You have not yet set up your credentials or your credentials are incorrect.
You must provide your credentials.
(aws-access-id): AKIAJOUAASEXAMPLE
(aws-secret-key): 5ZRIrtTM4ciIAvd4EXAMPLEDtm+PiPSzpoK
```

3. An Elastic Beanstalk application is a resource that contains a set of application versions (source), environments, and saved configurations that are associated with a single web application. Each time you deploy your source code to Elastic Beanstalk using the EB CLI, a new application version is created and added to the list.

```
Select an application to use
1) [ Create new Application ]
(default is 1): 1
```

4. The default application name is the name of the folder in which you run **eb init**. Enter any name that describes your project.

```
Enter Application Name
(default is "eb"): eb
Application eb has been created.
```

5. Select a platform that matches the language or framework that your web application is developed in. If you haven't started developing an application yet, choose a platform that you're interested in. You will see how to launch a sample application shortly, and you can always change this setting later.

```
Select a platform.
1) .NET Core on Linux
2) .NET on Windows Server
3) Docker
4) Go
5) Java
6) Node.js
7) PHP         <== select platform by number
8) Packer
9) Python
10) Ruby
11) Tomcat
(make a selection):7
```

6. Select a specific platform branch.

```
Select a platform branch.
1) PHP 8.4 running on 64bit Amazon Linux 2023
2) PHP 8.3 running on 64bit Amazon Linux 2023
3) PHP 8.2 running on 64bit Amazon Linux 2023
4) PHP 8.1 running on 64bit Amazon Linux 2023
5) PHP 8.1 running on 64bit Amazon Linux 2
(default is 1):1
```

7. Choose **yes** to assign an SSH key pair to the instances in your Elastic Beanstalk environment. This allows you to connect directly to them for troubleshooting.

```
Do you want to set up SSH for your instances?
(y/n): y
```

8. Choose an existing key pair or create a new one. To use **eb init** to create a new key pair, you must have **ssh-keygen** installed on your local machine and available from the command line. The EB CLI registers the new key pair with Amazon EC2 for you and stores the private key locally in a folder named `.ssh` in your user directory.

```
Select a keypair.
```

```
1) [ Create new KeyPair ]
(default is 1): 1
```

Your EB CLI installation is now configured and ready to use.

Advanced Configuration

- Ignoring files using .ebignore
- Using named profiles
- Deploying an artifact instead of the project folder
- Configuration settings and precedence
- Instance metadata

Ignoring files using .ebignore

You can tell the EB CLI to ignore certain files in your project directory by adding the file `.ebignore` to the directory. This file works like a `.gitignore` file. When you deploy your project directory to Elastic Beanstalk and create a new application version, the EB CLI doesn't include files specified by `.ebignore` in the source bundle that it creates.

If `.ebignore` isn't present, but `.gitignore` is, the EB CLI ignores files specified in `.gitignore`. If `.ebignore` is present, the EB CLI doesn't read `.gitignore`.

When `.ebignore` is present, the EB CLI doesn't use git commands to create your source bundle. This means that EB CLI ignores files specified in `.ebignore`, and includes all other files. In particular, it includes uncommitted source files.

Note

In Windows, adding `.ebignore` causes the EB CLI to follow symbolic links and include the linked file when creating a source bundle. This is a known issue and will be fixed in a future update.

Using named profiles

If you store your credentials as a named profile in a `credentials` or `config` file, you can use the `--profile` option to explicitly specify a profile. For example, the following command creates a new application using the `user2` profile.

```
$ eb init --profile user2
```

You can also change the default profile by setting the `AWS_EB_PROFILE` environment variable. When this variable is set, the EB CLI reads credentials from the specified profile instead of `default` or **eb-cli**.

Linux, macOS, or Unix

```
$ export AWS_EB_PROFILE=user2
```

Windows

```
> set AWS_EB_PROFILE=user2
```

Deploying an artifact instead of the project folder

You can tell the EB CLI to deploy a ZIP file or WAR file that you generate as part of a separate build process by adding the following lines to `.elasticbeanstalk/config.yml` in your project folder.

```
deploy:
  artifact: path/to/buildartifact.zip
```

If you configure the EB CLI in your Git repository, and you don't commit the artifact to source, use the `--staged` option to deploy the latest build.

```
~/eb$ eb deploy --staged
```

Configuration settings and precedence

The EB CLI uses a *provider chain* to look for AWS credentials in a number of different places, including system or user environment variables and local AWS configuration files.

The EB CLI looks for credentials and configuration settings in the following order:

1. **Command line options** – Specify a named profile by using `--profile` to override default settings.
2. **Environment variables** – `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
3. **The AWS credentials file** – Located at `~/.aws/credentials` on Linux and OS X systems, or at `C:\Users\USERNAME\.aws\credentials` on Windows systems. This file can contain multiple named profiles in addition to a default profile.
4. **The AWS CLI configuration file** – Located at `~/.aws/config` on Linux and OS X systems or `C:\Users\USERNAME\.aws\config` on Windows systems. This file can contain a default profile, named profiles, and AWS CLI–specific configuration parameters for each.
5. **Legacy EB CLI configuration file** – Located at `~/.elasticbeanstalk/config` on Linux and OS X systems or `C:\Users\USERNAME\.elasticbeanstalk\config` on Windows systems.
6. **Instance profile credentials** – These credentials can be used on Amazon EC2 instances with an assigned instance role, and are delivered through the Amazon EC2 metadata service. The instance profile must have permission to use Elastic Beanstalk.

If the credentials file contains a named profile with the name "eb-cli", the EB CLI will prefer that profile over the default profile. If no profiles are found, or a profile is found but does not have permission to use Elastic Beanstalk, the EB CLI prompts you to enter keys.

Instance metadata

To use the EB CLI from an Amazon EC2 instance, create a role that has access to the resources needed and assign that role to the instance when it is launched. Launch the instance and install the EB CLI by using `pip`.

```
~$ sudo pip install awsebcli
```

`pip` comes preinstalled on Amazon Linux.

The EB CLI reads credentials from the instance metadata. For more information, see Granting Applications that Run on Amazon EC2 Instances Access to AWS Resources in *IAM User Guide*.

Using the EB CLI with Git

The EB CLI provides a Git integration so you can associate code branches with specific environments in Elastic Beanstalk to organize your deploys.

To install Git and initialize your Git repository

1. Download the most recent version of Git by visiting http://git-scm.com.
2. Initialize your Git repository by typing the following:

   ```
   ~/eb$ git init
   ```

   EB CLI will now recognize that your application is set up with Git.
3. If you haven't already run **eb init**, do that now:

   ```
   ~/eb$ eb init
   ```

Associating Elastic Beanstalk environments with Git branches

You can associate a different environment with each branch of your code. When you checkout a branch, changes are deployed to the associated environment. For example, you can type the following to associate your production environment with your mainline branch, and a separate development environment with your development branch:

```
~/eb$ git checkout mainline
~/eb$ eb use prod
~/eb$ git checkout develop
~/eb$ eb use dev
```

Deploying changes

By default, the EB CLI deploys the latest commit in the current branch, using the commit ID and message as the application version label and description, respectively. If you want to deploy to your environment without committing, you can use the `--staged` option to deploy changes that have been added to the staging area.

To deploy changes without committing

1. Add new and changed files to the staging area:

   ```
   ~/eb$ git add .
   ```

2. Deploy the staged changes with **eb deploy**:

   ```
   ~/eb$ eb deploy --staged
   ```

If you have configured the EB CLI to deploy an artifact, and you don't commit the artifact to your git repository, use the `--staged` option to deploy the latest build.

Using Git submodules

Some code projects benefit from having Git submodules — repositories within the top-level repository. When you deploy your code using **eb create** or **eb deploy**, the EB CLI can include submodules in the application version zip file and upload them with the rest of the code.

You can control the inclusion of submodules by using the `include_git_submodules` option in the `global` section of the EB CLI configuration file, `.elasticbeanstalk/config.yml` in your project folder.

To include submodules, set this option to `true`:

```
global:
  include_git_submodules: true
```

When the `include_git_submodules` option is missing or set to `false`, EB CLI does not include submodules in the uploaded zip file.

See Git Tools - Submodules for more details about Git submodules.

Default behavior

When you run **eb init** to configure your project, the EB CLI adds the `include_git_submodules` option and sets it to `true`. This ensures that any submodules you have in your project are included in your deployments.

The EB CLI did not always support including submodules. To avoid an accidental and undesirable change to projects that had existed before we added submodule support, the EB CLI does not include submodules when the `include_git_submodules` option is missing. If you have one of these existing projects and you want to include submodules in your deployments, add the option and set it to `true` as explained in this section.

CodeCommit behavior

Elastic Beanstalk's integration with CodeCommit doesn't support submodules at this time. If you enabled your environment to integrate with CodeCommit, submodules are not included in your deployments.

Assigning Git tags to your application version

You can use a Git tag as your version label to identify what application version is running in your environment. For example, type the following:

```
~/eb$ git tag -a v1.0 -m "My version 1.0"
```

EB CLI command reference

You can use the Elastic Beanstalk command line interface (EB CLI) to perform a variety of operations to deploy and manage your Elastic Beanstalk applications and environments. The EB CLI integrates with Git if you want to deploy application source code that is under Git source control. For more information, see Setting up the EB command line interface (EB CLI) to manage Elastic Beanstalk and Using the EB CLI with Git.

Commands

- eb scale
- eb setenv
- eb ssh
- eb status
- eb swap
- eb tags
- eb terminate
- eb upgrade
- eb use

Common options

You can use the following options with all EB CLI commands.

Name	Description
`--debug`	Print information for debugging.
`-h, --help`	Show the Help message. Type: String Default: None
`--no-verify-ssl`	Skip SSL certificate verification. Use this option if you have issues using the CLI with a proxy.
`--profile`	Use a specific profile from your AWS credentials file.
`--quiet`	Suppress all output from the command.
`--region`	Use the specified region.
`-v, --verbose`	Display verbose information.

eb abort

Description

Cancels an upgrade when environment configuration changes to instances are still in progress.

> **Note**
>
> If you have more than two environments that are undergoing a update, you are prompted to select the name of the environment for which you want to roll back changes.

Syntax

eb abort

eb abort ***environment-name***

Options

Name	Description
Common options	

Output

The command shows a list of environments currently being updated and prompts you to choose the update that you want to abort. If only one environment is currently being updated, you do not need to specify the environment name. If successful, the command reverts environment configuration changes. The rollback process continues until all instances in the environment have the previous environment configuration or until the rollback process fails.

Example

The following example cancels the platform upgrade.

```
$ eb abort
Aborting update to environment "tmp-dev".
```

```
<list of events>
```

eb appversion

Description

The EB CLI `appversion` command manages your Elastic Beanstalk application versions. You can create a new version of the application without deploying, delete a version of the application, or create the application version lifecycle policy. If you invoke the command without any options, it enters the interactive mode.

Use the `--create` option to create a new version of the application.

Use the `--delete` option to delete a version of the application.

Use the `lifecycle` option to display or create the application version lifecycle policy. For more information, see the section called “Version lifecycle”.

Syntax

eb appversion

eb appversion [-c | --create]

eb appversion [-d | --delete] *`version-label`*

eb appversion lifecycle [-p | --print]

Options

Name	Description Type: String
-a *`application-name`* or --application_name *`application-name`*	The name of the application. If an application with the specified name isn't found, the EB CLI creates an application version for a new application. Only applicable with the `--create` option. Type: String

Name	Description Type: String
-c or --create	Create a new version of the application.
-d *version-label* or --delete *version-label*	Delete the version of the application that is labeled *version-label* .
`-l` *version_label* or `--label` *version_label*	Specify a label to use for the version that the EB CLI creates. If you don't use this option, the EB CLI generates a new unique label. If you provide a version label, make sure that it's unique. Only applicable with the `--create` option. Type: String
lifecycle	Invoke the default editor to create a new application version lifecycle policy. Use this policy to avoid reaching the applicati on version quota.
lifecycle -p or lifecycle --print	Display the current application lifecycle policy.

`-m "`*version_descriptio n*`"`
 	
or
 	
`--message "`*version_d escription*`"`
	The description for the application version. It's enclosed in double quotation marks.
	
	Only applicable with the `--create` option.
	
	Type: String

Name	Description Type: String

`-p`
 	
or
 	
`--process`
	Preprocess and validate the environment manifest and configuration files in the source bundle. Validating configura tion files can identify issues. We recommend you do this before deploying the application version to the environment.
	
	Only applicable with the `--create` option.

`--source codecommi t/` *`repository-name`*`/`*`branch-name`*	CodeCommit repository and branch. Only applicable with the `--create` option.
`--staged`	Use the files staged in the git index, instead of the HEAD commit, to create the application version. Only applicable with the `--create` option.
`--timeout` *`minutes`*	The number of minutes before the command times out. Only applicable with the `--create` option.
Common options	

Using the command interactively

If you use the command without any arguments, the output displays the versions of the application. They're listed in reverse chronological order, with the lastest version listed first. See the **Examples** section for examples of what the screen looks like. Note that the status line is displayed at the bottom. The status line displays context-sensitive information.

Press d to delete an application version, press `l` to manage the lifecycle policy for your application, or press q to quit without making any changes.

Note

If the version is deployed to any environment, you can't delete that version.

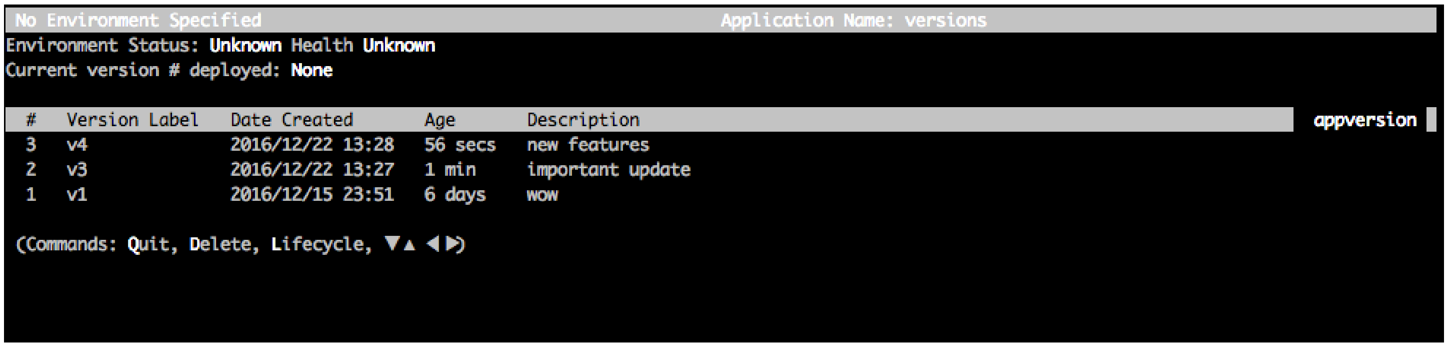
Output

The command with the `--create` option displays a message confirming that the application version was created.

The command with the `--delete` *`version-label`* option displays a message confirming that the application version was deleted.

Examples

The following example shows the interactive window for an application with no deployments.

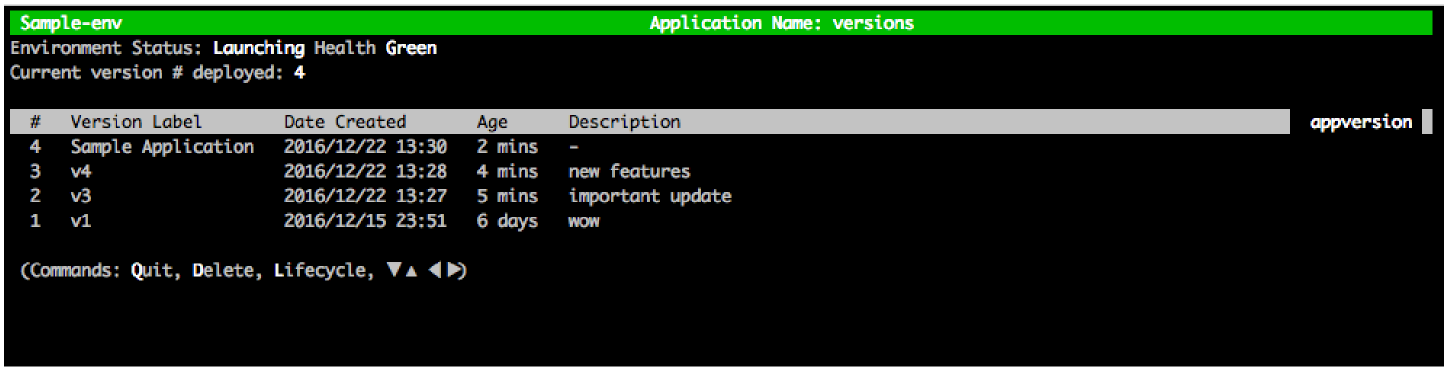


```
 No Environment Specified                                  Application Name: versions
Environment Status: Unknown Health Unknown
Current version # deployed: None

  #   Version Label   Date Created       Age       Description                                    appversion
  3   v4              2016/12/22 13:28   56 secs   new features
  2   v3              2016/12/22 13:27   1 min     important update
  1   v1              2016/12/15 23:51   6 days    wow

 (Commands: Quit, Delete, Lifecycle, ▼▲ ◀▶)
```

The following example shows the interactive window for an application with the fourth version, with version label **Sample Application**, deployed.



```
 Sample-env                                                Application Name: versions
Environment Status: Launching Health Green
Current version # deployed: 4

  #   Version Label        Date Created       Age      Description                                appversion
  4   Sample Application   2016/12/22 13:30   2 mins   -
  3   v4                   2016/12/22 13:28   4 mins   new features
  2   v3                   2016/12/22 13:27   5 mins   important update
  1   v1                   2016/12/15 23:51   6 days   wow

 (Commands: Quit, Delete, Lifecycle, ▼▲ ◀▶)
```

The following example shows the output from an **eb appversion lifecycle -p** command, where *`ACCOUNT-ID`* is the user's account ID:

```
Application details for: lifecycle
  Region: sa-east-1
  Description: Application created from the EB CLI using "eb init"
  Date Created: 2016/12/20 02:48 UTC
  Date Updated: 2016/12/20 02:48 UTC
  Application Versions: ['Sample Application']
  Resource Lifecycle Config(s):
    VersionLifecycleConfig:
      MaxCountRule:
        DeleteSourceFromS3: False
        Enabled: False
        MaxCount: 200
      MaxAgeRule:
        DeleteSourceFromS3: False
        Enabled: False
        MaxAgeInDays: 180
    ServiceRole: arn:aws:iam::ACCOUNT-ID:role/aws-elasticbeanstalk-service-role
```

eb clone

Description

Clones an environment to a new environment so that both have identical environment settings.

Note

By default, regardless of the solution stack version of the environment from which you create the clone, the **eb clone** command creates the clone environment with the most recent solution stack. You can suppress this by including the `--exact` option when you run the command.

Important

Cloned Elastic Beanstalk environments do not carry over the security groups for ingress, leaving the environment open to all internet traffic. You'll need to reestablish ingress security groups for the cloned environment.

You can see resources that may not be cloned by checking the drift status of your environment configuration. For more information, see Detect drift on an entire CloudFormation stack in the *AWS CloudFormation User Guide*.

Syntax

eb clone

eb clone *`environment-name`*

Options

Name	Description

`-n` *`string`*
 	
or
 	
`--clone_name` *`string`*
	Desired name for the cloned environment.

`-c` *`string`*
 	
or
 	
`--cname` *`string`*
	Desired CNAME prefix for the cloned environment.

`--envvars`	Environment properties in a comma-separated list with the format *`name`*`=`*`value`*. Type: String Constraints: • Key-value pairs must be separated by commas. • Keys and values can contain any alphabetic character in any language, any numeric character, white space, invisible separator, and the following symbols: _ . : / + \ - @ • Keys can contain up to 128 characters. Values can contain up to 256 characters. • Keys and values are case sensitive. • Values cannot match the environment name. • Values cannot include either `aws:` or `elasticbeanstalk:` .

Name	Description
	• The combined size of all environment properties cannot exceed 4096 bytes.
`--exact`	Prevents Elastic Beanstalk from updating the solution stack version for the new clone environment to the most recent version available (for the original environment's platform).
`--scale` *number*	The number of instances to run in the clone environment when it is launched.
`--tags` *name*=*value*	Tags for the resources in your environment in a comma-separated list with the format *name*=*value*.
`--timeout`	The number of minutes before the command times out.
Common options	

Output

If successful, the command creates an environment that has the same settings as the original environment or with modifications to the environment as specified by any **eb clone** options.

Example

The following example clones the specified environment.

```
$ eb clone
Enter name for Environment Clone
(default is tmp-dev-clone):
Enter DNS CNAME prefix
(default is tmp-dev-clone):
Environment details for: tmp-dev-clone
  Application name: tmp
  Region: us-west-2
  Deployed Version: app-141029_144740
  Environment ID: e-vjvrqnn5pv
  Platform: 64bit Amazon Linux 2014.09 v1.0.9 running PHP 5.5
  Tier: WebServer-Standard-1.0
  CNAME: tmp-dev-clone.elasticbeanstalk.com
  Updated: 2014-10-29 22:00:23.008000+00:00
Printing Status:
2018-07-11 21:04:20    INFO: createEnvironment is starting.
2018-07-11 21:04:21    INFO: Using elasticbeanstalk-us-west-2-888888888888 as Amazon S3
 storage bucket for environment data.
...
2018-07-11 21:07:10    INFO: Successfully launched environment: tmp-dev-clone
```

eb codesource

Description

Configures the EB CLI to deploy from a CodeCommit repository, or disables CodeCommit integration and uploads the source bundle from your local machine.

Note

Some AWS Regions don't offer CodeCommit. The integration between Elastic Beanstalk and CodeCommit doesn't work in these Regions.
For information about the AWS services offered in each Region, see Region Table.

Syntax

eb codesource

eb codesource codecommit

eb codesource local

Options

Name	Description
Common options	

Output

eb codesource prompts you to choose between CodeCommit integration and standard deployments.

eb codesource codecommit initiates interactive repository configuration for CodeCommit integration.

eb codesource local shows the original configuration and disables CodeCommit integration.

Examples

Use **eb codesource codecommit** to configure CodeCommit integration for the current branch.

```
~/my-app$ eb codesource codecommit
Select a repository
1) my-repo
2) my-app
3) [ Create new Repository ]
(default is 1): 1

Select a branch
1) mainline
2) test
3) [ Create new Branch with local HEAD ]
(default is 1): 1
```

Use **eb codesource local** to disable CodeCommit integration for the current branch.

```
~/my-app$ eb codesource local
Current CodeCommit setup:
  Repository: my-app
  Branch: mainline
Default set to use local sources
```

eb config

Description

Manages the active configuration settings and saved configurations of your environment. You can use this command to upload, download, or list the saved configurations of your environment. You can also use it to download, display, or update its active configuration settings.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command also changes the builder configuration settings. This is done based on the values that are set in `platform.yaml`.

> **Note**
>
> **eb config** doesn't show environment properties. To set environment properties that you can read from within your application, use **eb setenv** instead.

Syntax

The following are parts of the syntax that's used for the **eb config** command to work with the active configuration settings of your environment. For specific examples, see the Examples section later in this topic.

- **eb config** – Displays the active configuration settings of your environment in a text editor that you configured as the EDITOR environment variable. When you save changes to the file and close the editor, the environment is updated with the option settings that you saved in the file.

  > **Note**
  >
  > If you didn't configure an EDITOR environment variable, EB CLI displays your option settings in your default editor for YAML files.

- **eb config *`environment-name`*** – Displays and updates the configuration for the named environment. The configuration is either displayed in a text editor that you configured or your default editor YAML files.
- **eb config save** – Saves the active configuration settings for the current environment to `.elasticbeanstalk/saved_configs/` with the filename `[configuration-`

`name].cfg.yml`. By default, the EB CLI saves the configuration settings with a *`configuration-name`* based on the environment name. You can specify a different configuration name by including the `--cfg` option with your desired configuration name when you run the command.

You can tag your saved configuration using the `--tags` option.

- **eb config `--display`** – Writes an environment's active configuration settings to *stdout* instead of a file. By default this displays the configuration settings to the terminal.
- **eb config `--update` *`configuration_string`* `|` *`file_path`*** – Updates the active configuration settings for the current environment with the information that's specified in *`configuration_string`* or inside the file identified by *`file_path`*.

> **Note**
>
> The `--display` and `--update` options provide flexibility for reading and revising an environment's configuration settings programmatically.

The following describes the syntax for using the **eb config** command to work with saved configurations. For examples, see the Examples section later in this topic.

- **eb config get *`config-name`*** – Downloads the named saved configuration from Amazon S3.
- **eb config delete *`config-name`*** – Deletes the named saved configuration from Amazon S3. Also deletes it locally, if you already downloaded it.
- **eb config list** – Lists the saved configurations that you have in Amazon S3.
- **eb config put *`filename`*** – Uploads the named saved configuration to an Amazon S3 bucket. The *`filename`* must have the file extension `.cfg.yml`. To specify the file name without a path, you can save the file to the `.elasticbeanstalk` folder or to the `.elasticbeanstalk/saved_configs/` folder before you run the command. Alternatively, you can specify the *`filename`* by providing the full path.

Options

Name	Description
`--cfg` *config-name*	The name to use for a saved configuration. This option works with **eb config save** only.

`-d`
 	
or
 	
`--display`
	Displays the configuration settings for the current environment (writes to *stdout*).
	
	Use with the `--format` option to specify the output to be in JSON or YAML. If you don't specify, the output is in YAML format.
	
	This option only works if you use the **eb config** command without any of the other subcommands.

`-f` *format_type* or `--format` *format_type*	Specifies display format. Valid values are JSON or YAML. Defaults to YAML. This option works with the `--display` option only.
`--tags` *key1*=*value1*[,*ke*	Tags to add to your saved configuration. When specifying tags in the list, specify them as key=value pairs and separate each one with a comma. For more information, see Tagging saved configurations. This option works with **eb config save** only.
`--timeout` *timeout*	The number of minutes before the command times out.
`-u` *configuration_stri ng* | *file_path* or	Updates the active configuration settings for the current environment. This option only works if you use the **eb config** command without any of the other subcommands.

Name	Description

`--update` *`configuration_string`* | *`file_path`*	The *`configuration_string`* | *`file_path`* parameter is of the type string. The string provides the list of namespace s and corresponding options to add to, update, or remove from the configuration settings for your environment. Alternatively, the input string can represent a file that contains the same information.
	
	To specify a file name, the input string must follow the format `"file://<`*`path`*`><`*`filename`*`>"`. To specify the file name without a *`path`*, save the file to the folder where you run the command. Alternatively, specify the filename by providing the full path.
	
	The configuration information must meet the following conditions. At least one of the sections, **OptionSettings** or **OptionsToRemove**, is required. Use **OptionSettings** to add or change options. Use **OptionsToRemove** to remove options from a namespace. For specific examples, see the Examples section later in this topic.
	
	Example
	
	YAML Format
	```
	OptionSettings:
	  namespace1:
	    option-name-1:  option-value-1
	    option-name-2:  option-value-2
	    ...
	OptionsToRemove:
	  namespace1:
	    option-name-1
	    option-name-2
	    ...
	```
	Example
	
	JSON Format

Name	Description
	`{ "OptionSettings": { "namespace1": { "option-name-1": " option-value-1 ", "option-name-2": " option-value-2 ", ... } }, "OptionsToRemove": { "namespace1": { "option-name-1", "option-name-2", ... } } }`
Common options	

Output

If the **eb config** or **eb config *environment-name*** command is run successfully with no subcommands or options added, the command displays your current option settings in the text editor that you configured as the EDITOR environment variable. If you didn't configure an EDITOR environment variable, EB CLI displays your option settings in your default editor for YAML files.

When you save changes to the file and close the editor, the environment is updated with the option settings that you saved in the file. The following output is displayed to confirm the configuration update.

```
$ eb config myApp-dev
    Printing Status:
    2021-05-19 18:09:45    INFO    Environment update is starting.
    2021-05-19 18:09:55    INFO    Updating environment myApp-dev's configuration
 settings.
    2021-05-19 18:11:20    INFO    Successfully deployed new configuration to
 environment.
```

If the command runs successfully with the `--display` option, it displays the configuration settings for the current environment (writes to *stdout*).

If the command runs successfully with the `get` parameter, the command displays the location of the local copy that you downloaded.

If the command runs successfully with the `save` parameter, the command displays the location of the saved file.

Examples

This section describes how to change the text editor that you use to view and edit your option settings file.

For Linux and UNIX, the following example changes the editor to vim:

```
$ export EDITOR=vim
```

For Linux and UNIX, the following example changes the editor to whatever is installed at `/usr/bin/kate`.

```
$ export EDITOR=/usr/bin/kate
```

For Windows, the following example changes the editor to Notepad++.

```
> set EDITOR="C:\Program Files\Notepad++\Notepad++.exe
```

This section provides examples for the **eb config** command when it's run with subcommands.

The following example deletes the saved configuration named `app-tmp`.

```
$ eb config delete app-tmp
```

The following example downloads the saved configuration with the name app-tmp from your Amazon S3 bucket.

```
$ eb config get app-tmp
```

The following example lists the names of saved configurations that are stored in your Amazon S3 bucket.

```
$ eb config list
```

The following example uploads the local copy of the saved configuration named app-tmp to your Amazon S3 bucket.

```
$ eb config put app-tmp
```

The following example saves configuration settings from the current running environment. If you don't provide a name to use for the saved configuration, then Elastic Beanstalk names the configuration file according to the environment name. For example, an environment named *tmp-dev* would be called `tmp-dev.cfg.yml`. Elastic Beanstalk saves the file to the `/.elasticbeanstalk/saved_configs/` folder.

```
$ eb config save
```

In the following example, the `--cfg` option is used to save the configuration settings from the environment tmp-dev to a file called `v1-app-tmp.cfg.yml`. Elastic Beanstalk saves the file to the folder `/.elasticbeanstalk/saved_configs/`. If you don't specify an environment name, Elastic Beanstalk saves configuration settings from the current running environment.

```
$ eb config save tmp-dev --cfg v1-app-tmp
```

This section provides examples for the **eb config** command when it's run without subcommands.

The following command displays the option settings of your current environment in a text editor.

```
$ eb config
```

The following command displays the option settings for the *my-env* environment in a text editor.

```
$ eb config my-env
```

The following example displays the options settings for your current environment. It outputs in the YAML format because no specific format was specified with the `--format` option.

```
$ eb config --display
```

The following example updates the options settings for your current environment with the specifications in the file named `example.txt`. The file is in either the YAML or JSON format. The EB CLI automatically detects the file format.

- The Minsize option is set to 1 for the namespace `aws:autoscaling:asg`.
- The batch size for the namespace `aws:elasticbeanstalk:command` is set to 30%.
- It removes the option setting of *IdleTimeout: None* from the namespace `AWSEBV2LoadBalancer.aws:elbv2:loadbalancer`.

```
$ eb config --update "file://example.txt"
```

Example - filename: `example.txt` - YAML format

```
OptionSettings:
  'aws:elasticbeanstalk:command':
    BatchSize: '30'
    BatchSizeType: Percentage
  'aws:autoscaling:asg':
    MinSize: '1'
OptionsToRemove:
  'AWSEBV2LoadBalancer.aws:elbv2:loadbalancer':
    IdleTimeout
```

Example - filename: `example.txt` - JSON format

```
{
    "OptionSettings": {
        "aws:elasticbeanstalk:command": {
            "BatchSize": "30",
            "BatchSizeType": "Percentage"
        },
        "aws:autoscaling:asg": {
            "MinSize": "1"
        }
    },
    "OptionsToRemove": {
        "AWSEBV2LoadBalancer.aws:elbv2:loadbalancer": {
            "IdleTimeout"
        }
    }
}
```

The following examples update the options settings for your current environment. The command sets the Minsize option to 1 for the`aws:autoscaling:asg` namespace.

Note

These examples are specific to Windows PowerShell. They escape literal occurrences of the double-quote (") character by preceding it with a slash (\) character. Different operating systems and command-line environments might have different escape sequences. For this reason, we recommend using the file option that's shown in the previous examples. Specifying the configuration options in a file doesn't require escaping characters and is consistent across different operating systems.

The following example is in JSON format. The EB CLI detects if the format is in JSON or YAML.

```
PS C:\Users\myUser\EB_apps\myApp-env>eb config --update '{\"OptionSettings\":
{\"aws:autoscaling:asg\":{\"MaxSize\":\"1\"}}}'
```

The following example is in YAML format. To enter the YAML string in the correct format, the command includes spacing and end-of-line returns that are required in a YAML file.

- End each line with the "enter" or "return" key.
- Start the second line with two spaces, and start the third line with four spaces.

```
PS C:\Users\myUser\EB_apps\myApp-env>eb config --update 'OptionSettings:
>>  aws:autoscaling:asg:
>>    MinSize: \"1\"'
```

eb console

Description

Opens a browser to display the environment configuration dashboard in the Elastic Beanstalk Management Console.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command also displays the builder environment configuration, as specified in `platform.yaml`, in the Elastic Beanstalk Management Console.

Syntax

eb console

eb console *environment-name*

Options

Name	Description
Common options	

eb create

Description

Creates a new environment and deploys an application version to it.

Note

- To use **eb create** on a .NET application, you must create a deployment package as described in Creating a source bundle for a .NET application, then set up the CLI configuration to deploy the package as an artifact as described in Deploying an artifact instead of the project folder.
- Creating environments with the EB CLI requires a service role. You can create a service role by creating an environment in the Elastic Beanstalk console. If you don't have a service role, the EB CLI attempts to create one when you run `eb create`.

You can deploy the application version from a few sources:

- By default: From the application source code in the local project directory.
- Using the `--version` option: From an application version that already exists in your application.

- When your project directory doesn't have application code, or when using the `--sample` option: Deployed from a sample application, specific to your environment's platform.

Syntax

eb create

eb create *`environment-name`*

An environment name must be between 4 and 40 characters in length. It can only contain letters, numbers, and hyphens (-). An environment name can't begin or end with a hyphen.

If you include an environment name in the command, the EB CLI doesn't prompt you to make any selections or create a service role.

If you run the command without an environment name argument, it runs in an interactive flow, and prompts you to enter or select values for some settings. In this interactive flow, in case you are deploying a sample application, the EB CLI also asks you if you want to download this sample application to your local project directory. By downloading it, you can use the EB CLI with the new environment later to run operations that require the application's code, such as **eb deploy**.

Some interactive flow prompts are displayed only under certain conditions. For example, if you choose to use an Application Load Balancer, and your account has at least one sharable Application Load Balancer, Elastic Beanstalk displays a prompt that asks if you want to use a shared load balancer. If no sharable Application Load Balancer exists in your account, this prompt isn't displayed.

Options

None of these options are required. If you run **eb create** without any options, the EB CLI prompts you to enter or select a value for each setting.

Name	Description

`-d`
 	
or
 	
`--branch_default`
	Set the environment as the default environment for the current repository.

Name	Description
`--cfg` *`config-name`*	Use platform settings from a saved configuration in `.elasticbeanstalk/saved_configs/` or your Amazon S3 bucket. Specify the name of the file only, without the `.cfg.yml` extension.

`-c` *`subdomain-name`*
 	
or
 	
`--cname` *`subdomain-name`*
	The subdomain name to prefix the CNAME DNS entry that routes to your website.
	
	Type: String
	
	Default: The environment name

`-db`
 	
or
 	
`--database`
	Attaches a database to the environment. If you run **eb create** with the `--database` option, but without the `--database.username` and `--database.password` options, EB CLI prompts you for the database master user name and password.

`-db.engine` *`engine`*
 	
or
 	
`--database.engine` *`engine`*
	The database engine type. If you run **eb create** with this option, then EB CLI launches the environment with a database attached. This is the case even if you didn't run the command with the `--database` option.
	
	Type: String
	
	Valid values: `mysql`, `oracle-se1`, `postgres`, `sqlserver-ex`, `sqlserver-web`, `sqlserver-se`

Name	Description

`-db.i` *`instance_type`*
 	
or
 	
`--database.instance` *`instance_type`*
	The type of Amazon EC2 instance to use for the database. If you run **eb create** with this option, then EB CLI launches the environment with a database attached. This is the case even if you didn't run the command with the `--database` option.
	
	Type: String
	
	Valid values:
	
	Amazon RDS supports a standard set of DB instances. To select an appropriate DB instance for your DB engine, you must take into account some specific considerations. For more information, see DB instance classes in the *Amazon RDS User Guide*.

`-db.pass` *`password`*
 	
or
 	
`--database.password` *`password`*
	The password for the database. If you run **eb create** with this option, then EB CLI launches the environment with a database attached. This is the case even if you didn't run the command with the `--database` option.

Name	Description

`-db.size` *`number_of_gigabytes`*
 	
or
 	
`--database.size` *`number_of_gigabytes`*
	The number of gigabytes (GB) to allocate for database storage. If you run **eb create** with this option, then EB CLI launches the environment with a database attached. This is the case even if you didn't run the command with the `--database` option.
	
	Type: Number
	
	Valid values:
	
	• **MySQL** – `5` to `1024`. The default is `5`.
	• **Postgres** – `5` to `1024`. The default is `5`.
	• **Oracle** – `10` to `1024`. The default is `10`.
	• **Microsoft SQL Server Express Edition** – `30`.
	• **Microsoft SQL Server Web Edition** – `30`.
	• **Microsoft SQL Server Standard Edition** – `200`.

`-db.user` *`username`*
 	
or
 	
`--database.username` *`username`*
	The user name for the database. If you run **eb create** with this option, then EB CLI launches the environment with a database attached even if you didn't run the command with the `--database` option. If you run **eb create** with the `--database` option, but without the `--database.username` and `--database.password` options, then EB CLI prompts you for the master database user name and password.

`-db.version` *`version`*
 	
or
 	
`--database.version` *`version`*
	Used to specify the database engine version. If this flag is present, the environment will launch with a database with the specified version number, even if the `--database` flag isn't present.

Name	Description
`--elb-type` *`type`*	The load balancer type. Type: String Valid values: `classic`, `application` , `network` Default: `application`

`-es`
 	
or
 	
`--enable-spot`
	Enable Spot Instance requests for your environment. For more information, see Auto Scaling group.
	
	Related options:
	
	• `--instance-types`
	• `--on-demand-base-capacity`
	• `--on-demand-above-base-capacity`
	• `--spot-max-price`

`--env-group-suffix` *`groupname`*	The group name to append to the environment name. Only for use with Compose Environments.
`--envvars`	Environment properties in a comma-separated list with the format *`name`*`=`*`value`*. See Configuring environment properties (environment variables) for limits.

`-ip` *`profile_name`*
 	
or
 	
`--instance_profile` *`profile_name`*
	The instance profile with the IAM role with the temporary security credentials that your application needs to access AWS resources.

Name	Description

`-it`
 	
or
 	
`--instance-types` *`type1`*`[,`*`type2`*` ...]`
	A comma-separated list of Amazon EC2 instance types that you want your environment to use. If you don't specify this option, Elastic Beanstalk provides default instance types.
	
	For more information, see Amazon EC2 instances and Auto Scaling group.
	
	Important
	The EB CLI only applies this option to Spot Instances. Unless this option is used with the `--enable-spot` option, the EB CLI ignores it. To specify an instance type for an On-Demand Instance, use the `--intance-type` (no "s") option instead.

`-i`
 	
or
 	
`--instance_type`
	The Amazon EC2 instance type that you want your environment to use. If you don't specify this option, Elastic Beanstalk provides a default instance type.
	
	For more information, see Amazon EC2 instances.
	
	Important
	The EB CLI only applies this option to On-Demand Instances. Don't use this option with the `--enable-spot` option, because the EB CLI ignores it when you do so. To specify instance types for a Spot Instance, use the `--intance-types` (with an "s") option instead.

Name	Description

`-k` *`key_name`*
 	
or
 	
`--keyname` *`key_name`*
	The name of the Amazon EC2 key pair to use with the Secure Shell (SSH) client to securely log in to the Amazon EC2 instances that are running your Elastic Beanstalk application. If you include this option with the **eb create** command, the value you provide overwrites any key name that you might have specified with **eb init**.
	
	Valid values: An existing key name that's registered with Amazon EC2

`-im` *`number-of-instances`*
 	
or
 	
`--min-instances` *`number-of-instances`*
	The minimum number of Amazon EC2 instances that you require your environment to have.
	
	Type: Number (integer)
	
	Default: 1
	
	Valid values: 1 to `10000`

`-ix` *`number-of-instances`*
 	
or
 	
`--max-instances` *`number-of-instances`*
	The maximum number of Amazon EC2 instances you allow your environment to have.
	
	Type: Number (integer)
	
	Default: 4
	
	Valid values: 1 to `10000`

`--modules` *`component-a component-b`*	A list of component environments to create. This is only for use with Compose Environments.

Name	Description

`-sb`
 	
or
 	
`--on-demand-base-capacity`
	The minimum number of On-Demand Instances that your Auto Scaling group provisions before considering Spot Instances as your environment scales up.
	
	This option can only be specified with the `--enable-spot` option. For more information, see Auto Scaling group.
	
	Type: Number (integer)
	
	Default: `0`
	
	Valid values: `0` to `--max-instances` (when absent: `MaxSize` option in `aws:autoscaling:asg` namespace)

`-sp`
 	
or
 	
`--on-demand-above-base-capacity`
	The percentage of On-Demand Instances as part of additional capacity that your Auto Scaling group provisions that's more than the number of instances that's specified by the `--on-demand-base-capacity` option.
	
	This option can only be specified with the `--enable-spot` option. For more details, see Auto Scaling group.
	
	Type: Number (integer)
	
	Default: `0` for a single-instance environment; `70` for a load-balanced environment
	
	Valid values: `0` to `100`

Name	Description

`-p` *`platform-version`*
 	
or
 	
`--platform` *`platform-version`*
	The platform version to use. You can specify a platform, a platform and version, a platform branch, a solution stack name, or a solution stack ARN. For example:
	
	• php, PHP, `node.js` – The latest platform version for the specified platform
	• `php-7.2`, `"PHP 7.2"` – The recommended (typically latest) PHP 7.2 platform version
	• `"PHP 7.2 running on 64bit Amazon Linux"` – The recommended (typically latest) PHP platform version in this platform branch
	• `"64bit Amazon Linux 2017.09 v2.6.3 running PHP 7.1"` – The PHP platform version specified by this solution stack name
	• `"arn:aws:elasticbeanstalk:us-east-2::platform/PHP 7.1 running on 64bit Amazon Linux/2.6.3"` – The PHP platform version specified by this solution stack ARN
	
	Use `eb platform list` to get a list of available configurations.
	
	If you specify the `--platform` option, it overrides the value that was provided during `eb init`.

`-pr`
 	
or
 	
`--process`
	Preprocess and validate the environment manifest and configuration files in the source bundle. Validating configuration files can identify issues prior to deploying the application version to an environment.

Name	Description
`-r` *region* or `--region` *region*	The AWS Region where you want to deploy the applicati on. For the list of values you can specify for this option, see [AWS Elastic Beanstalk Endpoints and Quotas]() in the *AWS General Reference*.
`--sample`	Deploy the sample application to the new environment instead of the code in your repository.
`--scale` *number-of-instance s*	Launch with the specified number of instances
`--service-role` *servicerole*	Assign a non-default service role to the environment. **Note** Don't enter an ARN. Only enter the role name. Elastic Beanstalk prefixes the role name with the correct values to create the resulting ARN internally.

Name	Description

`-ls` *`load-balancer`*
 	
or
 	
`--shared-lb` *`load-balancer`*
	Configure the environment to use a shared load balancer. Provide the name or ARN of a sharable load balancer in your account—an Application Load Balancer that you explicitly created, not one created by another Elastic Beanstalk environment. For more information, see Shared Application Load Balancer.
	
	Parameter examples:
	
	• `FrontEndLB` – A load balancer name.
	• `arn:aws:elasticloadbalancing:us-east-2:123456789012:loadbalancer/app/FrontEndLB/0dbf78d8ad96abbc` – An Application Load Balancer ARN.
	
	You can specify this option only with `--elb-type application`. If you specify that option and don't specify `--shared-lb`, Elastic Beanstalk creates a dedicated load balancer for the environment.

`-lp` *`port`*
 	
or
 	
`--shared-lb-port` *`port`*
	The default listener port of the shared load balancer for this environment. Elastic Beanstalk adds a listener rule that routes all traffic from this listener to the default environment process. For more information, see Shared Application Load Balancer.
	
	Type: Number (integer)
	
	Default: `80`
	
	Valid values: Any integer that represents a listener port of the shared load balancer.

Name	Description
`--single`	Create the environment with a single Amazon EC2 instance and without a load balancer. **Warning** A single-instance environment isn't production ready. If the instance becomes unstable during deployment, or Elastic Beanstalk terminates and restarts the instance during a configuration update, your application can be unavailable for a period of time. Use single-instance environments for development, testing, or staging. Use load-balanced environments for production.

`-sm`
 	
or
 	
`--spot-max-price`
	The maximum price per unit hour, in US dollars, that you're willing to pay for a Spot Instance.
	
	This option can only be specified with the `--enable-spot` option. For more details, see Auto Scaling group.
	
	Type: Number (float)
	
	Default: The On-Demand price, for each instance type. The option's value in this case is `null`.
	
	Valid values: `0.001` to `20.0`
	
	For recommendations about maximum price options for Spot Instances, see Spot Instance pricing history in the *Amazon EC2 User Guide*.

`--tags` *`key1`*`=`*`value1`*`[,`*`key2`*`=`*`va`*	Tag the resources in your environment. Tags are specified as a comma-separated list of `key=value` pairs. For more information, see Tagging environments.

Name	Description

`-t worker`
 	
or
 	
`--tier worker`
	Create a worker environment. Omit this option to create a web server environment.

`--timeout` *`minutes`*	Set number of minutes before the command times out.
`--version` *`version_label`*	Specifies the application version that you want deployed to the environment instead of the application source code in the local project directory. Type: String Valid values: An existing application version label
`--vpc`	Configure a VPC for your environment. When you include this option, the EB CLI prompts you to enter all required settings prior to launching the environment.
`--vpc.dbsubnets` *`subnet1,subnet2`*	Specifies subnets for database instances in a VPC. Required when `--vpc.id` is specified.
`--vpc.ec2subnets` *`subnet1,subnet2`*	Specifies subnets for Amazon EC2 instances in a VPC. Required when `--vpc.id` is specified.
`--vpc.elbpublic`	Launches your Elastic Load Balancing load balancer in a public subnet in your VPC. You can't specify this option with the `--tier worker` or `--single` options.
`--vpc.elbsubnets` *`subnet1,subnet2`*	Specifies subnets for the Elastic Load Balancing load balancer in a VPC. You can't specify this option with the `--tier worker` or `--single` options.

Name	Description
`--vpc.id` *`ID`*	Launches your environment in the specified VPC.
`--vpc.publicip`	Launches your Amazon EC2 instances in a public subnet in your VPC. You can't specify this option with the `--tier worker` option.
`--vpc.securitygroups` *`securitygroup1,securitygroup2`*	Specifies security group IDs. Required when `--vpc.id` is specified.
Common options	

Output

If successful, the command prompts you with questions and then returns the status of the create operation. If there were problems during the launch, you can use the **eb events** operation to get more details.

If you enabled CodeBuild support in your application, **eb create** displays information from CodeBuild as your code is built. For information about CodeBuild support in Elastic Beanstalk, see Using the EB CLI with AWS CodeBuild.

Examples

The following example creates an environment in interactive mode.

```
$ eb create
Enter Environment Name
(default is tmp-dev): ENTER
Enter DNS CNAME prefix
(default is tmp-dev): ENTER
Select a load balancer type
1) classic
2) application
3) network
```

```
(default is 2): ENTER
Environment details for: tmp-dev
  Application name: tmp
  Region: us-east-2
  Deployed Version: app-141029_145448
  Environment ID: e-um3yfrzq22
  Platform: 64bit Amazon Linux 2014.09 v1.0.9 running PHP 5.5
  Tier: WebServer-Standard-1.0
  CNAME: tmp-dev.elasticbeanstalk.com
  Updated: 2014-10-29 21:54:51.063000+00:00
Printing Status:
...
```

The following example also creates an environment in interactive mode. In this example, your project directory doesn't have application code. The command deploys a sample application and downloads it to your local project directory.

```
$ eb create
Enter Environment Name
(default is tmp-dev): ENTER
Enter DNS CNAME prefix
(default is tmp-dev): ENTER
Select a load balancer type
1) classic
2) application
3) network
(default is 2): ENTER
NOTE: The current directory does not contain any source code. Elastic Beanstalk is
 launching the sample application instead.
Do you want to download the sample application into the current directory?
(Y/n): ENTER
INFO: Downloading sample application to the current directory.
INFO: Download complete.
Environment details for: tmp-dev
  Application name: tmp
  Region: us-east-2
  Deployed Version: Sample Application
  Environment ID: e-um3yfrzq22
  Platform: 64bit Amazon Linux 2014.09 v1.0.9 running PHP 5.5
  Tier: WebServer-Standard-1.0
  CNAME: tmp-dev.elasticbeanstalk.com
  Updated: 2017-11-08 21:54:51.063000+00:00
Printing Status:
```

```
...
```

The following command creates an environment without displaying any prompts.

```
$ eb create dev-env
Creating application version archive "app-160312_014028".
Uploading test/app-160312_014028.zip to S3. This may take a while.
Upload Complete.
Application test has been created.
Environment details for: dev-env
  Application name: test
  Region: us-east-2
  Deployed Version: app-160312_014028
  Environment ID: e-6fgpkjxyyi
  Platform: 64bit Amazon Linux 2015.09 v2.0.8 running PHP 5.6
  Tier: WebServer-Standard
  CNAME: UNKNOWN
  Updated: 2016-03-12 01:40:33.614000+00:00
Printing Status:
...
```

The following command creates an environment in a custom VPC.

```
$ eb create dev-vpc --vpc.id vpc-0ce8dd99 --vpc.elbsubnets subnet-
b356d7c6,subnet-02f74b0c --vpc.ec2subnets subnet-0bb7f0cd,subnet-3b6697c1 --
vpc.securitygroup sg-70cff265
Creating application version archive "app-160312_014309".
Uploading test/app-160312_014309.zip to S3. This may take a while.
Upload Complete.
Environment details for: dev-vpc
  Application name: test
  Region: us-east-2
  Deployed Version: app-160312_014309
  Environment ID: e-pqkcip3mns
  Platform: 64bit Amazon Linux 2015.09 v2.0.8 running Java 8
  Tier: WebServer-Standard
  CNAME: UNKNOWN
  Updated: 2016-03-12 01:43:14.057000+00:00
Printing Status:
...
```

eb deploy

Description

Deploys the application source bundle from the initialized project directory to the running application.

If git is installed, EB CLI uses the `git archive` command to create a `.zip` file from the contents of the most recent `git commit` command.

However, when `.ebignore` is present in your project directory, the EB CLI doesn't use git commands and semantics to create your source bundle. This means that EB CLI ignores files specified in `.ebignore`, and includes all other files. In particular, it includes uncommitted source files.

Note

You can configure the EB CLI to deploy an artifact from your build process instead of creating a ZIP file of your project folder. See Deploying an artifact instead of the project folder for details.

Syntax

eb deploy

eb deploy *environment-name*

Options

Name	Description
`-l` *version_label* or `--label` *version_label*	Specify a label to use for the version that the EB CLI creates. If the label has already been used, the EB CLI redeploys the previous version with that label. Type: String

Name	Description
`--env-group-suffix` *`groupname`*	Group name to append to the environment name. Only for use with Compose Environments.

`-m "`*`version_description`*`"`
 	
or
 	
`--message "`*`version_description`*`"`
	The description for the application version, enclosed in double quotation marks.
	
	Type: String

`--modules` *`component-a component-b`*	List of components to update. Only for use with Compose Environments.

`-p`
 	
or
 	
`--process`
	Preprocess and validate the environment manifest and configuration files in the source bundle. Validating configuration files can identify issues prior to deploying the application version to an environment.

`--source codecommit/`*`repository-name`*`/`*`branch-name`*	CodeCommit repository and branch.
`--staged`	Deploy files staged in the git index instead of the HEAD commit.
`--timeout` *`minutes`*	The number of minutes before the command times out.
`--version` *`version_label`*	An existing application version to deploy. Type: String
Common options	

Output

If successful, the command returns the status of the `deploy` operation.

If you enabled CodeBuild support in your application, **eb deploy** displays information from CodeBuild as your code is built. For information about CodeBuild support in Elastic Beanstalk, see Using the EB CLI with AWS CodeBuild.

Example

The following example deploys the current application.

```
$ eb deploy
2018-07-11 21:05:22    INFO: Environment update is starting.
2018-07-11 21:05:27    INFO: Deploying new version to instance(s).
2018-07-11 21:05:53    INFO: New application version was deployed to running EC2
 instances.
2018-07-11 21:05:53    INFO: Environment update completed successfully.
```

eb events

Description

Returns the most recent events for the environment.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command also returns the most recent events for the builder environment.

Syntax

eb events

eb events *environment-name*

Options

Name	Description

`-f`
 	
or
 	
`--follow`
	Streams events. To cancel, press CTRL+C.

Output

If successful, the command returns recent events.

Example

The following example returns the most recent events.

```
$ eb events
2014-10-29 21:55:39     INFO    createEnvironment is starting.
2014-10-29 21:55:40     INFO    Using elasticbeanstalk-us-west-2-111122223333 as Amazon
 S3 storage bucket for environment data.
2014-10-29 21:55:57     INFO    Created load balancer named: awseb-e-r-AWSEBLoa-
NSKUOK5X6Z9J
2014-10-29 21:56:16     INFO    Created security group named: awseb-e-rxgrhjr9bx-stack-
AWSEBSecurityGroup-1UUHU5LZ20ZY7
2014-10-29 21:57:18     INFO    Waiting for EC2 instances to launch. This may take a
 few minutes.
2014-10-29 21:57:18     INFO    Created Auto Scaling group named: awseb-e-rxgrhjr9bx-
stack-AWSEBAutoScalingGroup-1TE320ZCJ9RPD
2014-10-29 21:57:22     INFO    Created Auto Scaling group policy named:
 arn:aws:autoscaling:us-east-2:11122223333:scalingPolicy:2cced9e6-859b-421a-
be63-8ab34771155a:autoScalingGroupName/awseb-e-rxgrhjr9bx-stack-
AWSEBAutoScalingGroup-1TE320ZCJ9RPD:policyName/awseb-e-rxgrhjr9bx-stack-
AWSEBAutoScalingScaleUpPolicy-1I2ZSNVU4APRY
2014-10-29 21:57:22     INFO    Created Auto Scaling group policy named:
 arn:aws:autoscaling:us-east-2:11122223333:scalingPolicy:1f08b863-
bf65-415a-b584-b7fa3a69a0d5:autoScalingGroupName/awseb-e-rxgrhjr9bx-stack-
AWSEBAutoScalingGroup-1TE320ZCJ9RPD:policyName/awseb-e-rxgrhjr9bx-stack-
AWSEBAutoScalingScaleDownPolicy-1E3G7PZKZPSOG
2014-10-29 21:57:25     INFO    Created CloudWatch alarm named: awseb-e-rxgrhjr9bx-
stack-AWSEBCloudwatchAlarmLow-VF5EJ549FZBL
2014-10-29 21:57:25     INFO    Created CloudWatch alarm named: awseb-e-rxgrhjr9bx-
stack-AWSEBCloudwatchAlarmHigh-LA9YEW3O6WJO
2014-10-29 21:58:50     INFO    Added EC2 instance 'i-c7ee492d' to Auto ScalingGroup
 'awseb-e-rxgrhjr9bx-stack-AWSEBAutoScalingGroup-1TE320ZCJ9RPD'.
2014-10-29 21:58:53     INFO    Successfully launched environment: tmp-dev
2014-10-29 21:59:14     INFO    Environment health has been set to GREEN
2014-10-29 21:59:43     INFO    Adding instance 'i-c7ee492d' to your environment.
```

eb health

Description

Returns the most recent health for the environment.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command also returns the most recent health for the builder environment.

Syntax

eb health

eb health *environment-name*

Options

Name	Description

`-r`
 	
or
 	
`--refresh`
	Show health information interactively and update every 10 seconds as new information is reported.

`--mono`	Don't display color in output.

Output

If successful, the command returns recent health.

Example

The following example returns the most recent health information for a Linux environment.

```
~/project $ eb health
 elasticBeanstalkExa-env                                  Ok
 2015-07-08 23:13:20
WebServer
 Ruby 2.1 (Puma)
```

```
   total      ok    warning  degraded  severe    info   pending  unknown
     5        5        0        0        0        0        0        0

   instance-id   status     cause
                                             health
     Overall     Ok
   i-d581497d    Ok
   i-d481497c    Ok
   i-136e00c0    Ok
   i-126e00c1    Ok
   i-8b2cf575    Ok

   instance-id   r/sec    %2xx   %3xx   %4xx   %5xx      p99      p90      p75     p50
    p10                            requests
     Overall     671.8   100.0    0.0    0.0    0.0    0.003    0.002    0.001   0.001
  0.000
   i-d581497d    143.0    1430      0      0      0    0.003    0.002    0.001   0.001
  0.000
   i-d481497c    128.8    1288      0      0      0    0.003    0.002    0.001   0.001
  0.000
   i-136e00c0    125.4    1254      0      0      0    0.004    0.002    0.001   0.001
  0.000
   i-126e00c1    133.4    1334      0      0      0    0.003    0.002    0.001   0.001
  0.000
   i-8b2cf575    141.2    1412      0      0      0    0.003    0.002    0.001   0.001
  0.000

   instance-id   type       az   running     load 1  load 5      user%  nice%  system%
  idle%   iowait%                             cpu
   i-d581497d    t2.micro   1a   12 mins        0.0    0.04        6.2    0.0      1.0
  92.5       0.1
   i-d481497c    t2.micro   1a   12 mins       0.01    0.09        5.9    0.0      1.6
  92.4       0.1
   i-136e00c0    t2.micro   1b   12 mins       0.15    0.07        5.5    0.0      0.9
  93.2       0.0
   i-126e00c1    t2.micro   1b   12 mins       0.17    0.14        5.7    0.0      1.4
  92.7       0.1
   i-8b2cf575    t2.micro   1c   1 hour        0.19    0.08        6.5    0.0      1.2
  92.1       0.1

   instance-id   status     id   version              ago
                                     deployments
   i-d581497d    Deployed   1    Sample Application   12 mins
   i-d481497c    Deployed   1    Sample Application   12 mins
   i-136e00c0    Deployed   1    Sample Application   12 mins
   i-126e00c1    Deployed   1    Sample Application   12 mins
   i-8b2cf575    Deployed   1    Sample Application   1 hour
```

eb init

Description

Sets default values for Elastic Beanstalk applications created with EB CLI by prompting you with a series of questions.

Note

The values you set with **eb init** apply to the current directory and repository on the current computer.
The command creates an Elastic Beanstalk application in your account. To create an Elastic Beanstalk environment, run **eb create** after running **eb init**.

Syntax

eb init

eb init *`application-name`*

Options

If you run **eb init** without specifying the `--platform` option, the EB CLI prompts you to enter a value for each setting.

Note

To use **eb init** to create a new key pair, you must have `ssh-keygen` installed on your local machine and available from the command line.

Name	Description	

`-i`
 	
`--interactive`
	Forces EB CLI to prompt you to provide a value for every **eb init** command option.
	
	Note
	
	The `init` command prompts you to provide values for **eb init** command options that do not have a (default) value. After the first time you run the **eb init** command in a directory , EB CLI might not prompt you about any command options. Therefore, use the `--interactive` option when you want to change a setting that you previously set.	

`-k` *`keyname`*
 	
`--keyname` *`keyname`*
	The name of the Amazon EC2 key pair to use with the Secure Shell (SSH) client to securely log in to the Amazon EC2 instances running your Elastic Beanstalk application.	

`--modules` *`folder-1 folder-2`*	List of child directories to initialize. Only for use with Compose Environments.	

`-p` *`platform-version`*
 	
`--platform` *`platform-version`*
	The platform version to use. You can specify a platform, a platform and version, a platform branch, a solution stack name, or a solution stack ARN. For example:
	
	• php, PHP, `node.js` – The latest platform version for the specified platform
	• `php-7.2`, `"PHP 7.2"` – The recommended (typically latest) PHP 7.2 platform version
	• `"PHP 7.2 running on 64bit Amazon Linux"` – The recommended (typically latest) PHP platform version in this platform branch	

Name	Description	
	• `"64bit Amazon Linux 2017.09 v2.6.3 running PHP 7.1"` – The PHP platform version specified by this solution stack name • `"arn:aws:elasticbeanstalk:us-east-2::platform/PHP 7.1 running on 64bit Amazon Linux/2.6.3"` – The PHP platform version specified by this solution stack ARN Use `eb platform list` to get a list of available configurations. Specify the `--platform` option to skip interactive configuration. **Note** When you specify this option, then EB CLI does not prompt you for values for any other options. Instead, it assumes default values for each option. You can specify options for anything for which you do not want to use default values.	
`--source codecommit/ *repository-name*/*branch-name*`	CodeCommit repository and branch.	
`--tags *key1*=*value1*`	Tag your application. Tags are specified as a comma-separated list of `key=value` pairs. For more details, see Tagging applications.	
Common options		

CodeBuild support

If you run **eb init** in a folder that contains a buildspec.yml file, Elastic Beanstalk parses the file for an **eb_codebuild_settings** entry with options specific to Elastic Beanstalk. For information about CodeBuild support in Elastic Beanstalk, see Using the EB CLI with AWS CodeBuild.

Output

If successful, the command guides you through setting up a new Elastic Beanstalk application through a series of prompts.

Example

The following example request initializes EB CLI and prompts you to enter information about your application. Replace *placeholder* text with your own values.

```
$ eb init -i
Select a default region
1) us-east-1 : US East (N. Virginia)
2) us-west-1 : US West (N. California)
3) us-west-2 : US West (Oregon)
4) eu-west-1 : Europe (Ireland)
5) eu-central-1 : Europe (Frankfurt)
6) ap-south-1 : Asia Pacific (Mumbai)
7) ap-southeast-1 : Asia Pacific (Singapore)
...
(default is 3): 3

Select an application to use
1) HelloWorldApp
2) NewApp
3) [ Create new Application ]
(default is 3): 3

Enter Application Name
(default is "tmp"):
Application tmp has been created.

It appears you are using PHP. Is this correct?
(y/n): y
```

```
Select a platform branch.
1) PHP 7.2 running on 64bit Amazon Linux
2) PHP 7.1 running on 64bit Amazon Linux (Deprecated)
3) PHP 7.0 running on 64bit Amazon Linux (Deprecated)
4) PHP 5.6 running on 64bit Amazon Linux (Deprecated)
5) PHP 5.5 running on 64bit Amazon Linux (Deprecated)
6) PHP 5.4 running on 64bit Amazon Linux (Deprecated)
(default is 1): 1
Do you want to set up SSH for your instances?
(y/n): y

Select a keypair.
1) aws-eb
2) [ Create new KeyPair ]
(default is 2): 1
```

eb labs

Description

Subcommands of **eb labs** support work-in-progress or experimental functionality. These commands may be removed or reworked in future versions of the EB CLI and are not guaranteed to be forward compatible.

For a list of available subcommands and descriptions, run **eb labs --help**.

eb list

Description

Lists all environments in the current application or all environments in all applications, as specified by the `--all` option.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command also lists the builder environments.

Syntax

eb list

Options

Name	Description

`-a`
 	
or
 	
`--all`
	Lists all environments from all applications.

`-v`
 	
or
 	
`--verbose`
	Provides more detailed information about all environments, including instances.

Common options	

Output

If successful, the command returns a list of environment names in which your current environment is marked with an asterisk (*).

Example 1

The following example lists your environments and indicates that tmp-dev is your default environment.

```
$ eb list
* tmp-dev
```

Example 2

The following example lists your environments with additional details.

```
$ eb list --verbose
Region: us-west-2
Application: tmp
    Environments: 1
```

```
        * tmp-dev : ['i-c7ee492d']
```

eb local

Description

Use **eb local run** to run your application's containers locally in Docker. Check the application's container status with **eb local status**. Open the application in a web browser with **eb local open**. Retrieve the location of the application's logs with **eb local logs**.

eb local setenv and **eb local printenv** let you set and view environment variables that are provided to the Docker containers that you run locally with **eb local run**.

You must run all **eb local** commands in the project directory of a Docker application that has been initialized as an EB CLI repository by using **eb init**.

Note

Use **eb local** on a local computer running Linux or macOS. The command doesn't support Windows.
Before using the command on macOS, install Docker for Mac, and ensure that boot2docker isn't installed (or isn't in the execution path). The **eb local** command tries to use boot2docker if it's present, but doesn't work well with it on macOS.

Syntax

eb local run

eb local status

eb local open

eb local logs

eb local setenv

eb local printenv

Options

eb local run

Name	Description
`--envvars` *`key1=value1,key2=value2`*	Sets environment variables that the EB CLI will pass to the local Docker containers. In multicontainer environments, all variables are passed to all containers.
`--port` *`hostport`*	Maps a port on the host to the exposed port on the container . If you don't specify this option, the EB CLI uses the same port on both host and container. This option works only with Docker platform applications. It doesn't apply to the Multicontainer Docker platform.
Common options	

eb local status

eb local open

eb local logs

eb local setenv

eb local printenv

Name	Description
Common options	

Output

eb local run

Status messages from Docker. Remains active as long as application is running. Press **Ctrl+C** to stop the application.

eb local status

The status of each container used by the application, running or not.

eb local open

Opens the application in a web browser and exits.

eb local logs

The location of the logs generated in your project directory by applications running locally under **eb local run**.

eb local setenv

None

eb local printenv

The name and values of environment variables set with **eb local setenv**.

Examples

eb local run

```
~/project$ eb local run
Creating elasticbeanstalk_phpapp_1...
Creating elasticbeanstalk_nginxproxy_1...
Attaching to elasticbeanstalk_phpapp_1, elasticbeanstalk_nginxproxy_1
phpapp_1     | [23-Apr-2015 23:24:25] NOTICE: fpm is running, pid 1
phpapp_1     | [23-Apr-2015 23:24:25] NOTICE: ready to handle connections
```

eb local status

View the status of your local containers:

```
~/project$ eb local status
Platform: 64bit Amazon Linux 2014.09 v1.2.1 running Multi-container Docker 1.3.3
 (Generic)
Container name: elasticbeanstalk_nginxproxy_1
Container ip: 127.0.0.1
Container running: True
Exposed host port(s): 80
Full local URL(s): 127.0.0.1:80

Container name: elasticbeanstalk_phpapp_1
Container ip: 127.0.0.1
Container running: True
Exposed host port(s): None
Full local URL(s): None
```

eb local logs

View the log path for the current project:

```
~/project$ eb local logs
Elastic Beanstalk will write logs locally to /home/user/project/.elasticbeanstalk/logs/
local.
Logs were most recently created 3 minutes ago and written to /home/user/
project/.elasticbeanstalk/logs/local/150420_234011665784.
```

eb local setenv

Set environment variables for use with **eb local run**.

```
~/project$ eb local setenv PARAM1=value
```

Print environment variables set with **eb local setenv**.

```
~/project$ eb local printenv
Environment Variables:
PARAM1=value
```

eb logs

Description

The **eb logs** command has two distinct purposes: to enable or disable log streaming to CloudWatch Logs, and to retrieve instance logs or CloudWatch Logs logs. With the `--cloudwatch-logs` (`-cw`) option, the command enables or disables log streaming. Without this option, it retrieves logs.

When retrieving logs, specify the `--all`, `--zip`, or `--stream` option to retrieve complete logs. If you don't specify any of these options, Elastic Beanstalk retrieves tail logs.

The command processes logs for the specified or default environment. Relevant logs vary by container type. If the root directory contains a `platform.yaml` file specifying a custom platform, this command also processes logs for the builder environment.

For more information, see the section called "CloudWatch Logs".

Syntax

To enable or disable log streaming to CloudWatch Logs:

```
eb logs --cloudwatch-logs [enable | disable] [--cloudwatch-log-source instance |
 environment-health | all] [environment-name]
```

To retrieve instance logs:

```
eb logs [-all | --zip | --stream] [--cloudwatch-log-source instance] [--
instance instance-id] [--log-group log-group] [environment-name]
```

To retrieve environment health logs:

```
eb logs [-all | --zip | --stream] --cloudwatch-log-source environment-health
 [environment-name]
```

Options

Name	Description

`-cw [enable | disable]`
 	
or
 	
`--cloudwatch-logs [enable | disable]`
	Enables or disables log streaming to CloudWatch Logs. If no argument is supplied, log streaming is enabled. If the `--cloudwatch-log-source` (`-cls`) option isn't specified in addition, instance log streaming is enabled or disabled.

`-cls instance | environment-health | all`	Specifies the source of logs when working with CloudWatch Logs. With the enable or disable form of the command, these are the logs for which to enable or disable CloudWatch Logs

Name	Description
or `--cloudwatch-log-source instance | environment-health | all`	streaming. With the retrieval form of the command, these are the logs to retrieve from CloudWatch Logs. Valid values: • With `--cloudwatch-logs` (enable or disable) – `instance | environment-health | all` • Without `--cloudwatch-logs` (retrieve) – `instance | environment-health` Value meanings: • `instance` (default) – Instance logs • `environment-health` – Environment health logs (supported only when enhanced health is enabled in the environment) • `all` – Both log sources

`-a`
 	
or
 	
`--all`
	Retrieves complete logs and saves them to the `.elasticbeanstalk/logs` directory.

`-z`
 	
or
 	
`--zip`
	Retrieves complete logs, compresses them into a `.zip` file, and then saves the file to the `.elasticbeanstalk/logs` directory.

`--stream`	Streams (continuously outputs) complete logs. With this option, the command keeps running until you interrupt it (press **Ctrl+C**).

Name	Description

`-i` *`instance-id`*
 	
or
 	
`--instance` *`instance-id`*
	Retrieves logs for the specified instance only.

`-g` *`log-group`*
 	
or
 	
`--log-group` *`log-group`*
	Specifies the CloudWatch Logs log group from which to retrieve logs. The option is valid only when instance log streaming to CloudWatch Logs is enabled.
	
	If instance log streaming is enabled, and you don't specify the `--log-group` option, the default log group is one of the following:
	
	• Amazon Linux 2 – `/aws/elasticbeanstalk/`*`environment-name`*`/var/log/eb-engine.log`
	• Windows platforms – `/aws/elasticbeanstalk/`*`environment-name`*`/EBDeploy-Log`
	• Amazon Linux AMI (AL1) – `/aws/elasticbeanstalk/`*`environment-name`*`/var/log/eb-activity.log`
	
	Note
	On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information about migrating to a current and fully supported Amazon Linux 2023 platform branch, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.
	
	For information about the log group corresponding to each log file, see How Elastic Beanstalk sets up CloudWatch Logs.

Name	Description
Common options	

Output

By default, displays the logs directly in the terminal. Uses a paging program to display the output. Press **Q** or **q** to exit.

With `--stream`, shows existing logs in the terminal and keeps running. Press **Ctrl+C** to exit.

With `--all` and `--zip`, saves the logs to local files and displays the file location.

Examples

The following example enables instance log streaming to CloudWatch Logs.

```
$ eb logs -cw enable
Enabling instance log streaming to CloudWatch for your environment
After the environment is updated you can view your logs by following the link:
https://console.aws.amazon.com/cloudwatch/home?region=us-east-1#logs:prefix=/aws/
elasticbeanstalk/environment-name/
Printing Status:
2018-07-11 21:05:20    INFO: Environment update is starting.
2018-07-11 21:05:27    INFO: Updating environment environment-name's configuration
 settings.
2018-07-11 21:06:45    INFO: Successfully deployed new configuration to environment.
```

The following example retrieves instance logs into a `.zip` file.

```
$ eb logs --zip
Retrieving logs...
Logs were saved to /home/workspace/environment/.elasticbeanstalk/logs/150622_173444.zip
```

eb migrate

Description

Migrates Internet Information Services (IIS) sites and applications from a Windows server to Elastic Beanstalk. The command packages your applications, preserves their configurations, and deploys them to a new Elastic Beanstalk environment.

For more information about migrating your IIS sites and applications, see *Migrating IIS applications*.

Note

Before using this command, ensure your system meets these requirements:

- Internet Information Services (IIS) version 7.0 or later
- Web Deploy 3.6 or later installed
- Administrative privileges on the Windows server
- AWS credentials configured with appropriate permissions
- Your source server has outbound internet access to AWS services.

The following steps summarize the migration process:

1. Discover IIS sites and their configurations.
2. Package application content and configuration.
3. Create Elastic Beanstalk environment and application.
4. Deploy the application with preserved settings.

The command creates migration artifacts in a structured directory as shown in the following listing:

```
C:\migration_workspace\
### .\migrations\latest\
    ### upload_target.zip
    ### upload_target\
        ### [SiteName].zip                      # One ZIP per default application of
 IIS site
        ### [SiteName-ApplicationName].zip      # One ZIP per additional application
        ### aws-windows-deployment-manifest.json
        ### ebmigrateScripts\
            ### site_installer.ps1              # Site installation scripts
            ### permission_handler.ps1          # Permission management
            ### other helper scripts
```

Use **eb migrate cleanup** to manage these artifacts.

Syntax

eb migrate *[options]*

eb migrate explore *[options]*

eb migrate cleanup *[options]*

When run without arguments, **eb migrate** operates in non-interactive mode. To execute it in the interactive mode, run **eb migrate --interactive**.

The interactive mode command prompts for the following information:

- Selection of IIS sites to migrate
- Environment and application names
- Platform version selection
- Instance type and other configuration options

Subcommands

explore

The **eb migrate explore** subcommand examines your IIS server and lists available sites.

Use this command to display the following information:

- View all IIS sites on the server
- With `--verbose`, inspect detailed configuration including:
  - Site bindings and ports
  - Application pools
  - Virtual directories and their physical paths
  - Authentication settings

```
PS C:\migrations_workspace >  eb migrate explore
Default Web Site
Site2
site3
router
```

```
PS C:\migrations_workspace >  eb migrate explore --verbose
1: Default Web Site:
  - Bindings:
    - *:8083:
  - Application '/':
    - Application Pool: DefaultAppPool
    - Enabled Protocols: http
    - Virtual Directories:
      - /:
        - Physical Path: C:\inetpub\wwwroot
        - Logon Method: ClearText
  - Application '/dotnet-6-0':
    - Application Pool: DefaultAppPool
    - Enabled Protocols: http
    - Virtual Directories:
      - /:
        - Physical Path: C:\inetpub\AspNetCoreWebApps\CoreWebApp-6-0
        - Logon Method: ClearText
  - Application '/dotnet-8-0':
    - Application Pool: DefaultAppPool
    - Enabled Protocols: http
    - Virtual Directories:
      - /:
        - Physical Path: C:\inetpub\AspNetCoreWebApps\CoreWebApp-8-0
        - Logon Method: ClearText
2: Site2:
  - Bindings:
    - *:8081:
...
```

cleanup

The **eb migrate cleanup** subcommand manages migration artifacts with the following actions:

- Preserving the most recent successful migration in `./migrations/latest`
- Removing older migration directories

- Maintaining critical configuration files

```
PS C:\migrations_workspace >   eb migrate cleanup
Are you sure you would like to cleanup older artifacts within `./migrations/`? (y/N):
```

Use `--force` to skip confirmation prompts during cleanup.

```
PS C:\migrations_workspace >   eb migrate cleanup --force
```

Options

None of these options are required. If you run **eb migrate** without any options, the EB CLI will execute in the non-interactive mode. With **eb migrate --interactive**, the EB CLI prompts you to enter or select a value for required settings.

Name	Description

`-a` *`application-name`*
 	
or
 	
`--application-name` *`application-name`*
	Name for the new Elastic Beanstalk application.
	
	Type: String
	
	Default: EBMigratedApp

`--archive` *`directory-or-zip`*	The directory or ZIP file containing source code previousl y generated by **eb migrate --archive-only**. Use this option to deploy a previously created migration package. Example: `--archive .\migrations\latest \upload_target` or `--archive .\migrations \latest\upload_target.zip`

`-ao`
 	
or
 	
`--archive-only`
	Create only the destination archive directory without deployment.

Name	Description
	The resulting directory can be manually deployed using **eb migrate** with the `archive` option, or **eb deploy**.

`-c` *`subdomain-name`*
 	
or
 	
`--cname` *`subdomain-name`*
	The subdomain name to prefix the CNAME DNS entry for your migrated application.
	
	Type: String
	
	Default: The environment name

`-cf`
 	
or
 	
`--copy-firewall-config`
	Copy source server firewall configuration to the destinati on for all HTTP ports with active bindings.
	
	Creates corresponding security group rules in AWS.

`-es` *`snapshot-id`* [*`snapshot-id`* ...] or `--ebs-snapshots` *`snapshot-id`* [*`snapshot-id`* ...]	Comma-separated list of Amazon EBS snapshot IDs to associate with the environment. Example: `--ebs-snapshots snap-1234567890abcdef0, snap-0987654321fedcba1`
`--encrypt-ebs-volumes`	Enforce encryption for all new Amazon EBS volumes. **Important** This is an account-wide setting that affects all future Amazon EBS volume creation.

Name	Description

`-e` *`environment-name`*
 	
or
 	
`--environment-name` *`environment-name`*
	Name for the new Elastic Beanstalk environment.
	
	Type: String
	
	Default: EBMigratedEnv
	
	Constraints: Must be between 4 and 40 characters in length. Can only contain letters, numbers, and hyphens. Cannot start or end with a hyphen.

`--force`	Skip confirmation prompts during operations. When used with **cleanup** subcommand, removes migration artifacts without confirmation.

`-ip` *`profile-name`*
 	
or
 	
`--instance-profile` *`profile-name`*
	Instance Profile to associate with the environment's Amazon EC2 instances.
	
	If not specified, creates a default instance profile with permissions to access Elastic Beanstalk resources. For more information, see the section called "Instance profile".

`-i` *`instance-type`*
 	
or
 	
`--instance-type` *`instance-type`*
	The Amazon EC2 instance type for your Elastic Beanstalk environment.
	
	Type: String
	
	Default: c5.2xlarge
	
	For available instance types, see Amazon EC2 instance types in the *Amazon EC2 User Guide*.

`-in`
 	
or
 	
`--interactive`
	Force interactive mode for the migration process.
	
	Prompts for configuration values even when defaults are available.

Name	Description

`-k` *`key-name`*
 	
or
 	
`--keyname` *`key-name`*
	Amazon EC2 key pair to enable RDP access to environme nt instances.
	
	Useful for investigating instance-level issues not visible in logs.
	
	Valid values: An existing key pair name registered with Amazon EC2

`-p` *`platform-version`*
 	
or
 	
`--platform` *`platform-version`*
	Elastic Beanstalk platform runtime for the environme nt. If not specified, automatically detected from host Windows Server version.
	
	Example: `"64bit Windows Server 2016 v2.16.2 running IIS 10.0"`
	
	For a list of available platform versions, use eb platform list.

Name	Description
`--remote`	Indicates to execute the migration in remote mode. This option allows execution from a bastion host, which connects to the target server that contains the application and configurations to be migrated to Elastic Beanstalk. Running from the bastion server, `eb migrate` discovers configurations, stages migration logic on the bastion host, then deploys your application to a new Elastic Beanstalk environment. This option eliminates the need to install the EB CLI and Python on the Windows server that you need to migrate. You install Python and the EB CLI on a bastion host instead, where you run the **eb migrate** command with the `--remote` option. Use the `--target-ip` option to specify the host with the IIS configurations to migrate. Must be used with `--target-ip` , `--username` , and `--password` .
`--target-ip` *`ip-address`*	Public IP address of the remote Windows machine that contains the IIS servers to be migrated. Required when using `--remote`. Can only be specified when using `--remote`.
`--username` *`username`*	Username of the user profile to access the remote Windows machine that contains the IIS servers to be migrated. Required when using `--remote`. Can only be specified when using `--remote`.

Name	Description
`--password` *`password`*	Password of the user profile to access the remote Windows machine that contains the IIS servers to be migrated. Required when using `--remote`. Can only be specified when using `--remote`.

`-sr` *`role-name`*
 	
or
 	
`--service-role` *`role-name`*
	IAM service role for Elastic Beanstalk to manage related AWS services.
	
	If not specified, creates a default service role with necessary permissions. For more information, see the section called "Service role".
	
	Note
	Specify only the role name, not the full ARN. Elastic Beanstalk automatically creates the complete ARN.

`-s` *`site-names`*
 	
or
 	
`--sites` *`site-names`*
	Comma-separated list of IIS sites to migrate. If not specified, migrates all available sites on the server.
	
	Example: `--sites "Default Web Site,Intranet,API"`

`--ssl-certificates` *`certificate-arn`* [,*`certificate-arn`* ...]	Comma-separated list of ACM SSL certificate ARNs to associate with the Application Load Balancer. Required when migrating sites with HTTPS bindings. Example: `--ssl-certificates arn:aws:acm:region:account:certificate/certificate-id`

Name	Description

`-t key1=value1[,key2=value2 ..`
 	
or
 	
`--tags key1=value1[,key2=va`
	Comma-separated list of `key=value` pairs to tag new resources in your environment: Environment, Elastic Beanstalk application, Application version.
	
	For more information, see Tagging environments.

`--verbose`	Show detailed information during migration process. When used with **explore** subcommand, displays comprehensive site configuration details.

Name	Description

`-vpc` *`config-file-or-string`*
 	
or
 	
`--vpc-config` *`config-file-or-string`*
	VPC configuration for the environment, specified either as a JSON file path or a JSON string.
	
	Configuration must include:
	
	`{`
	`    "id": "vpc-1234567890abcdef0",`
	`    "publicip": "true|false",`
	`    "elbscheme": "public|private",`
	`    "ec2subnets": ["subnet-a1b2c3d4", "subnet-e5f6g7h8"],`
	`    "securitygroups": "sg-123456,sg-789012",`
	`    "elbsubnets": ["subnet-a1b2c3d4", "subnet-e5f6g7h8"]`
	`}`
	
	• `id`: (Required) VPC identifier
	• `publicip`: Whether to assign public IPs to instances
	• `elbscheme` : Load balancer scheme (public or private)
	• `ec2subnets` : List of subnet IDs for EC2 instances
	• `securitygroups` : Comma-separated security group IDs
	• `elbsubnets` : List of subnet IDs for the load balancer
	
	Important
	
	The migration will ignore any existing VPC settings from the source environment when you specify the `--vpc-config` *parameter.* When you use this parameter, the migration will only use the VPC settings specified in the configuration file that you're passing in. Using this parameter overrides the default behavior of

Name	Description
	discovering the source instance's VPC configura tion or using the default VPC.
Common options	

Output

The command provides status updates throughout the migration process:

1. VPC configuration detection (when running on an EC2 instance)
2. Source bundle generation progress for each site
3. Environment creation status
4. Deployment progress

If successful, displays the new environment's details including:

- Environment name and ID
- Application name
- Region
- Platform version
- Environment CNAME

For issues during migration, use the **eb events** and **eb health** commands to get detailed information.

Examples

Basic Usage

Basic migration in interactive mode:

```
PS C:\migrations_workspace >  eb migrate
Identifying VPC configuration of this EC2 instance (i-0123456789abcdef0):
  id: vpc-1234567890abcdef0
  publicip: true
  elbscheme: public
  ec2subnets: subnet-123,subnet-456,subnet-789
  securitygroups: sg-123,sg-456
  elbsubnets: subnet-123,subnet-456,subnet-789

Using .\migrations\latest to contain artifacts for this migration run.
Generating source bundle for sites, applications, and virtual directories...
  Default Web Site/ -> .\migrations\latest\upload_target\DefaultWebSite.zip

Creating application version
Creating environment

Environment details for: EBMigratedEnv
  Application name: EBMigratedApp
  Region: us-west-2
  Deployed Version: app-230320_153045
  Environment ID: e-abcdef1234
  Platform: 64bit Windows Server 2019 v2.7.0 running IIS 10.0
  Tier: WebServer-Standard-1.0
  CNAME: ebmigratedenv.us-west-2.elasticbeanstalk.com
  Updated: 2023-03-20 15:30:45
```

Migrating specific sites with custom configuration:

```
PS C:\migrations_workspace >  eb migrate `
    --sites "Default Web Site,InternalAPI" `
    --application-name "CorporateApp" `
    --environment-name "Production" `
    --instance-type "c5.xlarge" `
    --tags "Environment=Production,Team=WebOps" `
    --copy-firewall-config
```

Creating migration archive without deployment:

```
PS C:\migrations_workspace >  eb migrate --archive-only
Using .\migrations\latest to contain artifacts for this migration run.
Generating source bundle for sites, applications, and virtual directories...
  Default Web Site/ -> .\migrations\latest\upload_target\DefaultWebSite.zip

Generated destination archive directory at .\migrations\latest\upload_target
You can execute `eb init` and `eb create` from this directory to deploy to EB.
```

Advanced Configuration Examples

Migration with custom VPC configuration using a JSON file:

```
PS C:\migrations_workspace >  cat vpc-config.json
{
    "id": "vpc-1234567890abcdef0",
    "publicip": "false",
    "elbscheme": "internal",
    "ec2subnets": [
        "subnet-private1",
        "subnet-private2"
    ],
    "securitygroups": [
        "sg-app",
        "sg-database",
        "sg-monitoring"
    ],
    "elbsubnets": [
        "subnet-private1",
        "subnet-private2"
    ]
}

PS C:\migrations_workspace eb migrate `
    --sites "InternalAPI" `
    --vpc-config vpc-config.json `
    --instance-type "r5.xlarge" `
    --tags "Environment=Internal,Security=High"
```

Migrating sites with SSL certificates and host headers:

```
PS C:\migrations_workspace >  eb migrate `
    --sites "SecurePortal" `
    --ssl-certificates "arn:aws:acm:region:account:certificate/
cert1,arn:aws:acm:region:account:certificate/cert2" `
    --verbose
INFO: Detected HTTPS bindings:
  - www.example.com:443
  - api.example.com:443

INFO: Configuring Application Load Balancer with SSL certificates
INFO: Creating host-based routing rules:
  - www.example.com -> target group 1
  - api.example.com -> target group 2
```

Migration with EBS snapshot configuration:fo

```
PS C:\migrations_workspace >  eb migrate `
    --sites "Default Web Site" `
    --ebs-snapshots "snap-1234567890abcdef0" "snap-0987654321fedcba1" `
    --encrypt-ebs-volumes
Using .\migrations\latest to contain artifacts for this migration run.
INFO: Enabling EBS encryption for all new volumes in us-west-2
INFO: Configuring environment with specified EBS snapshots
```

Security Configuration Examples

Handling sites with complex firewall rules:

```
PS C:\migrations_workspace >  eb migrate `
    --sites "Default Web Site,ReportingService" `
    --copy-firewall-config `
    --verbose
INFO: Detected the following Windows Firewall rules:
  - Allow Web Traffic (TCP 80, 443)
  - Allow Reporting Traffic (TCP 8081)
INFO: Creating corresponding security group rules
```

Migration with custom IAM roles:

```
PS C:\migrations_workspace >  eb migrate `
    --sites "SecureApp" `
    --instance-profile "CustomInstanceProfile" `
    --service-role "CustomServiceRole"
```

Remote Execution Examples

Migrating IIS applications from a remote Windows server:

```
PS C:\migrations_workspace >  eb migrate `
    --remote `
    --target-ip "192.0.2.10" `
    --username "administrator" `
    --password "YourPassword123" `
    --application-name "RemoteApp" `
    --environment-name "RemoteEnv"
INFO: Establishing SSH connection to remote host 192.0.2.10...
INFO: Connection established
INFO: Discovering IIS sites on remote host...
INFO: Found 2 sites: Default Web Site, API
INFO: Extracting site configurations...
INFO: Generating source bundle for sites, applications, and virtual directories...
  Default Web Site/ -> .\migrations\latest\upload_target\DefaultWebSite.zip
  API/ -> .\migrations\latest\upload_target\API.zip

Creating application version
Creating environment

Environment details for: RemoteEnv
  Application name: RemoteAppstage mi
  Region: us-west-2
  Deployed Version: app-230320_153045
  Environment ID: e-abcdef1234
  Platform: 64bit Windows Server 2019 v2.7.0 running IIS 10.0
  Tier: WebServer-Standard-1.0
  CNAME: remoteenv.us-west-2.elasticbeanstalk.com
  Updated: 2023-03-20 15:30:45
```

Remote migration with specific site selection:

```
PS C:\migrations_workspace >  eb migrate `
    --remote `
    --target-ip "192.0.2.10" `
    --username "administrator" `
    --password "YourPassword123" `
    --sites "API" `
    --instance-type "c5.large"
```

eb open

Description

Opens the public URL of your website in the default browser.

Syntax

eb open

eb open ***environment-name***

Options

Name	Description
Common options	

Output

The command **eb open** does not have output. Instead, it opens the application in a browser window.

eb platform

Description

This command supports two different workspaces:

Platform

Use this workspace to manage custom platforms.

Environment

Use this workspace to select a default platform or show information about the current platform.

Elastic Beanstalk provides the shortcut **ebp** for **eb platform**.

Note

Windows PowerShell uses **ebp** as a command alias. If you're running the EB CLI in Windows PowerShell, use the long form of this command — **eb platform**.

Using eb platform for custom platforms

Lists the versions of the current platform and enables you to manage custom platforms.

Syntax

eb platform create [*version*] [*options*]

eb platform delete [*version*] [*options*]

eb platform events [*version*] [*options*]

eb platform init [*platform*] [*options*]

eb platform list [*options*]

eb platform logs [*version*] [*options*]

eb platform status [*version*] [*options*]

eb platform use [*platform*] [*options*]

Options

Name	Description
`create [version] [options]`	Build a new version of the platform. Learn more.
`delete version [options]`	Delete a platform version. Learn more.
`events [version] [options]`	Display the events from a platform version. Learn more.
`init [platform] [options]`	Initialize a platform repository. Learn more.
`list [options]`	List the versions of the current platform. Learn more.

Name	Description
`logs [version] [options]`	Display logs from the builder environment for a platform version. Learn more.
`status [version] [options]`	Display the status of the a platform version. Learn more.
`use [platform] [options]`	Select a different platform from which new versions are built. Learn more.
Common options	

Common options

All **eb platform** commands include the following common options.

Name	Description
`-h` OR `--help`	Shows a help message and exits.
`--debug`	Shows additional debugging output.
`--quiet`	Suppresses all output.
`-v` OR `--verbose`	Shows additional output.
`--profile PROFILE`	Uses the specified *PROFILE* from your credentials.
`-r REGION` OR	Use the region *REGION*.

Name	Description
`--region` *`REGION`*	
`--no-verify-ssl`	Do not verify AWS SSL certificates.

Eb platform create

Builds a new version of the platform and returns the ARN for the new version. If there is no builder environment running in the current region, this command launches one. The *`version`* and increment options (`-M`, `-m`, and `-p`) are mutually exclusive.

Options

Name	Description
`version`	If *`version`* isn't specified, creates a new version based on the most-recent platform with the patch version (N in n.n.N) incremented.

`-M`
 	
OR
 	
`--major-increment`
	Increments the major version number (the N in N.n.n).

`-m`
 	
OR
 	
`--minor-increment`
	Increments the minor version number (the N in n.N.n).

`-p`
 	
OR
 	
`--patch-increment`
	Increments the patch version number (the N in n.n.N).

`-i` *`INSTANCE_TYPE`* OR	Use *`INSTANCE_TYPE`* as the instance type, such as **`t1.micro`**.

Name	Description
--instance-type *INSTANCE_ TYPE*	
`-ip` *INSTANCE_PROFILE* OR `--instance-profile` *INSTANCE_PROFILE*	Use *INSTANCE_PROFILE* as the instance profile when creating AMIs for a custom platform. If the `-ip` option isn't specified, creates the instance profile `aws-elasticbeanstalk-custom-platforme-ec2-role` and uses it for the custom platform.
`--tags` *key1*=*value1*[,*ke*	Tags are specified as a comma-separated list of `key=value` pairs.
`--timeout` *minutes*	Set number of minutes before the command times out.
`--vpc.id` *VPC_ID*	The ID of the VPC in which Packer builds.
`--vpc.subnets` *VPC_SUBNETS*	The VPC subnets in which Packer builds.
`--vpc.publicip`	Associates public IPs to EC2 instances launched.

Eb platform delete

Delete a platform version. The version isn't deleted if an environment is using that version.

Options

Name	Description
version	The version to delete. This value is required.
`--cleanup`	Remove all platform versions in the `Failed` state.
`--all-platforms`	If `--cleanup` is specified, remove all platform versions in the `Failed` state for all platforms.
`--force`	Do not require confirmation when deleting a version.

Eb platform events

Display the events from a platform version. If *`version`* is specified, display the events from that version, otherwise display the events from the current version.

Options

Name	Description
`version`	The version for which events are displayed. This value is required.

`-f`
 	
OR
 	
`--follow`
	Continue to display events as they occur.

Eb platform init

Initialize a platform repository.

Options

Name	Description
`platform`	The name of the platform to initialize. This value is required, unless `-i` (interactive mode) is enabled.

`-i`
 	
OR
 	
`--interactive`
	Use interactive mode.

`-k` *`KEYNAME`*
 	
OR
 	
`--keyname` *`KEYNAME`*
	The default EC2 key name.

You can run this command in a directory that has been previously initialized, although you cannot change the workspace type if run in a directory that has been previously initialized.

To re-initialize with different options, use the `-i` option.

Eb platform list

List the versions of the platform associated with a workspace (directory) or a region.

The command returns different results depending on the type of workspace you run it in, as follows:

- In a platform workspace (a directory initialized by `eb platform init`), the command returns a list of all platform versions of the custom platform defined in the workspace. Add the `--all-platforms` or `--verbose` option to get a list of all platform versions of all custom platforms your account has in the region associated with the workspace.
- In an application workspace (a directory initialized by `eb init`), the command returns a list of all platform versions, both for platforms managed by Elastic Beanstalk and for your account's custom platforms. The list uses short platform version names, and some platform version variants might be combined. Add the `--verbose` option to get a detailed list with full names and all variants listed separately.
- In an uninitialized directory, the command only works with the `--region` option. It returns a list of all Elastic Beanstalk-managed platform versions supported in the region. The list uses short platform version names, and some platform version variants might be combined. Add the `--verbose` option to get a detailed list with full names and all variants listed separately.

Options

Name	Description

`-a`
 	
OR
 	
`--all-platforms`
	Valid only in an initialized workspace (a directory initialized by `eb platform init` or `eb init`). Lists the platform versions of all custom platforms associated with your account.

`-s` *STATUS* OR	List only the platforms matching *STATUS*: • Ready

Name	Description
`--status STATUS`	• Failed • Deleting • Creating

Eb platform logs

Display logs from the builder environment for a platform version.

Options

Name	Description
`version`	The version of the platform for which logs are displayed. If omitted, display logs from the current version.
`--stream`	Stream deployment logs that were set up with CloudWatch.

Eb platform status

Display the status of the a platform version.

Options

Name	Description
`version`	The version of the platform for which the status is retrieved. If omitted, display the status of the current version.

Eb platform use

Select a different platform from which new versions are built.

Options

Name	Description
platform	Specifies *platform* as the active version for this workspace. This value is required.

Using eb platform for environments

Lists supported platforms and enables you to set the default platform and platform version to use when you launch an environment. Use **eb platform list** to view a list of all supported platforms. Use **eb platform select** to change the platform for your project. Use **eb platform show** to view your project's selected platform.

Syntax

eb platform list

eb platform select

eb platform show

Options

Name	Description
`list`	List the version of the current platform.
`select`	Select the default platform.
`show`	Show information about the current platform.

Example 1

The following example lists the names of all configurations for all platforms that Elastic Beanstalk supports.

```
$ eb platform list
```

```
docker-1.5.0
glassfish-4.0-java-7-(preconfigured-docker)
glassfish-4.1-java-8-(preconfigured-docker)
go-1.3-(preconfigured-docker)
go-1.4-(preconfigured-docker)
iis-7.5
iis-8
iis-8.5
multi-container-docker-1.3.3-(generic)
node.js
php-5.3
php-5.4
php-5.5
python
python-2.7
python-3.4
python-3.4-(preconfigured-docker)
ruby-1.9.3
ruby-2.0-(passenger-standalone)
ruby-2.0-(puma)
ruby-2.1-(passenger-standalone)
ruby-2.1-(puma)
ruby-2.2-(passenger-standalone)
ruby-2.2-(puma)
tomcat-6
tomcat-7
tomcat-7-java-6
tomcat-7-java-7
tomcat-8-java-8
```

Example 2

The following example prompts you to choose from a list of platforms and the version that you want to deploy for the specified platform.

```
$ eb platform select
Select a platform.
1) PHP
2) Node.js
3) IIS
4) Tomcat
5) Python
6) Ruby
```

```
7) Docker
8) Multi-container Docker
9) GlassFish
10) Go
(default is 1): 5

Select a platform version.
1) Python 2.7
2) Python
3) Python 3.4 (Preconfigured - Docker)
```

Example 3

The following example shows information about the current default platform.

```
$ eb platform show
Current default platform: Python 2.7
New environments will be running:  64bit Amazon Linux 2014.09 v1.2.0 running Python 2.7

Platform info for environment "tmp-dev":
Current: 64bit Amazon Linux 2014.09 v1.2.0 running Python
Latest:  64bit Amazon Linux 2014.09 v1.2.0 running Python
```

eb printenv

Description

Prints all the environment properties in the command window.

Syntax

eb printenv

eb printenv *environment-name*

Options

Name	Description
Common options	

Output

If successful, the command returns the status of the `printenv` operation.

Example

The following example prints environment properties for the specified environment.

```
$ eb printenv
Environment Variables:
     PARAM1 = Value1
```

eb restore

Description

Rebuilds a terminated environment, creating a new environment with the same name, ID, and configuration. The environment name, domain name, and application version must be available for use in order for the rebuild to succeed.

Syntax

eb restore

eb restore *environment_id*

Options

Name	Description
Common options	

Output

The EB CLI displays a list of terminated environments that are available to restore.

Example

```
$ eb restore
```

```
Select a terminated environment to restore

  #   Name          ID             Application Version      Date Terminated        Ago

  3   gamma         e-s7mimej8e9   app-77e3-161213_211138   2016/12/14 20:32 PST   13
 mins
  2   beta          e-sj28uu2wia   app-77e3-161213_211125   2016/12/14 20:32 PST   13
 mins
  1   alpha         e-gia8mphu6q   app-77e3-161213_211109   2016/12/14 16:21 PST   4
 hours

 (Commands: Quit, Restore, # #)

Selected environment alpha
Application:    scorekeep
Description:    Environment created from the EB CLI using "eb create"
CNAME:          alpha.h23tbtbm92.us-east-2.elasticbeanstalk.com
Version:        app-77e3-161213_211109
Platform:       64bit Amazon Linux 2016.03 v2.1.6 running Java 8
Terminated:     2016/12/14 16:21 PST
Restore this environment? [y/n]: y

2018-07-11 21:04:20    INFO: restoreEnvironment is starting.
2018-07-11 21:04:39    INFO: Created security group named: sg-e2443f72
...
```

eb scale

Description

Scales the environment to always run on a specified number of instances, setting both the minimum and maximum number of instances to the specified number.

Syntax

eb scale ***`number-of-instances`***

eb scale ***`number-of-instances environment-name`***

Options

Name	Description
--timeout	The number of minutes before the command times out.
Common options	

Output

If successful, the command updates the number of minimum and maximum instances to run to the specified number.

Example

The following example sets the number of instances to 2.

```
$ eb scale 2
2018-07-11 21:05:22    INFO: Environment update is starting.
2018-07-11 21:05:27    INFO: Updating environment tmp-dev's configuration settings.
2018-07-11 21:08:53    INFO: Added EC2 instance 'i-5fce3d53' to Auto Scaling Group
 'awseb-e-2cpfjbra9a-stack-AWSEBAutoScalingGroup-7AXY7U13ZQ6E'.
2018-07-11 21:08:58    INFO: Successfully deployed new configuration to environment.
2018-07-11 21:08:59    INFO: Environment update completed successfully.
```

eb setenv

Description

Sets environment properties for the default environment.

Syntax

eb setenv *key*=*value*

You can include as many properties as you want, but the total size of all properties cannot exceed 4096 bytes. You can delete a variable by leaving the value blank. See Configuring environment properties (environment variables) for limits.

Note

If the `value` contains a special character, you must escape that character by preceding it with a \ character.

Options

Name	Description
--timeout	The number of minutes before the command times out.
Common options	

Output

If successful, the command displays that the environment update succeeded.

Example

The following example sets the environment variable ExampleVar.

```
$ eb setenv ExampleVar=ExampleValue
2018-07-11 21:05:25    INFO: Environment update is starting.
2018-07-11 21:05:29    INFO: Updating environment tmp-dev's configuration settings.
2018-07-11 21:06:50    INFO: Successfully deployed new configuration to environment.
2018-07-11 21:06:51    INFO: Environment update completed successfully.
```

The following command sets multiple environment properties. It adds the environment property named `foo` and sets its value to `bar`, changes the value of the `JDBC_CONNECTION_STRING` property, and deletes the `PARAM4` and `PARAM5` properties.

```
$ eb setenv foo=bar JDBC_CONNECTION_STRING=hello PARAM4= PARAM5=
```

eb ssh

Description

Note

This command does not work with environments running Windows Server instances.

Connect to a Linux Amazon EC2 instance in your environment using Secure Shell (SSH). If an environment has multiple running instances, EB CLI prompts you to specify which instance you want to connect to. To use this command, SSH must be installed on your local machine and available from the command line. Private key files must be located in a folder named `.ssh` under your user directory, and the EC2 instances in your environment must have public IP addresses.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command also connects to instances in the custom environment.

SSH keys

If you have not previously configured SSH, you can use the EB CLI to create a key when running **eb init**. If you have already run **eb init**, run it again with the `--interactive` option and select **Yes** and **Create New Keypair** when prompted to set up SSH. Keys created during this process will be stored in the proper folder by the EB CLI.

This command temporarily opens port 22 in your environment's security group for incoming traffic from 0.0.0.0/0 (all IP addresses) if no rules for port 22 are already in place. If you have configured your environment's security group to open port 22 to a restricted CIDR range for increased security, the EB CLI will respect that setting and forgo any changes to the security group. To override this behavior and force the EB CLI to open port 22 to all incoming traffic, use the `--force` option.

See EC2 security groups for information on configuring your environment's security group.

Syntax

eb ssh

eb ssh ***`environment-name`***

Options

Name	Description

`-i`
 	
or
 	
`--instance`
	Specifies the instance ID of the instance to which you connect. We recommend that you use this option.

`-n`
 	
or
 	
`--number`
	Specify the instance to connect to by number.

`-o`
 	
or
 	
`--keep_open`
	Leave port 22 open on the security group after the SSH session ends.

`--command`	Execute a shell command on the specified instance instead of starting an SSH session.
`--custom`	Specify an SSH command to use instead of 'ssh -i keyfile'. Do not include the remote user and hostname.
`--setup`	Change the key pair assigned to the environment's instances (requires instances to be replaced).
`--force`	Open port 22 to incoming traffic from 0.0.0.0/0 in the environment's security group, even if the security group is already configured for SSH. Use this option if your environment's security group is configured to open port 22 to a restricted CIDR range that does not include the IP address that you are trying to connect from.
`--timeout` *`minutes`*	Set number of minutes before the command times out.

Name	Description
	Can only be used with the `--setup` argument.
Common options	

Output

If successful, the command opens an SSH connection to the instance.

Example

The following example connects you to the specified environment.

```
$ eb ssh
Select an instance to ssh into
1) i-96133799
2) i-5931e053
(default is 1): 1
INFO: Attempting to open port 22.
INFO: SSH port 22 open.
The authenticity of host '54.191.45.125 (54.191.45.125)' can't be established.
RSA key fingerprint is ee:69:62:df:90:f7:63:af:52:7c:80:60:1b:3b:51:a9.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '54.191.45.125' (RSA) to the list of known hosts.

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2014.09-release-notes/
No packages needed for security; 1 packages available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-8-185 ~]$ ls
[ec2-user@ip-172-31-8-185 ~]$ exit
logout
Connection to 54.191.45.125 closed.
INFO: Closed port 22 on ec2 instance security group
```

eb status

Description

Provides information about the status of the environment.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command also provides information about the builder environment.

Syntax

eb status

eb status *`environment-name`*

Options

Name	Description

`-v`
 	
or
 	
`--verbose`
	Provides more information about individual instances, such as their status with the Elastic Load Balancing load balancer.

Common options	

Output

If successful, the command returns the following information about the environment:

- Environment name
- Application name
- Deployed application version
- Environment ID
- Platform
- Environment tier

- CNAME
- Time the environment was last updated
- Status
- Health

If you use verbose mode, EB CLI also provides you with the number of running Amazon EC2 instances.

Example

The following example shows the status for the environment tmp-dev.

```
$ eb status
Environment details for: tmp-dev
  Application name: tmp
  Region: us-west-2
  Deployed Version: None
  Environment ID: e-2cpfjbra9a
  Platform: 64bit Amazon Linux 2014.09 v1.0.9 running PHP 5.5
  Tier: WebServer-Standard-1.0
  CNAME: tmp-dev.elasticbeanstalk.com
  Updated: 2014-10-29 21:37:19.050000+00:00
  Status: Launching
  Health: Grey
```

eb swap

Description

Swaps the environment's CNAME with the CNAME of another environment (for example, to avoid downtime when you update your application version).

Note

If you have more than two environments, you are prompted to select the name of the environment that is currently using your desired CNAME from a list of environments. To suppress this, you can specify the name of the environment to use by including the `-n` option when you run the command.

Syntax

eb swap

eb swap *environment-name*

> **Note**
>
> The *environment-name* is the environment for which you want a different CNAME. If you don't specify *environment-name* as a command line parameter when you run **eb swap**, EB CLI updates the CNAME of the default environment.

Options

Name	Description

`-n`
 	
or
 	
`--destination_name`
	Specifies the name of the environment with which you want to swap CNAMEs. If you run **eb swap** without this option, then EB CLI prompts you to choose from a list of your environments.

Common options	

Output

If successful, the command returns the status of the `swap` operation.

Examples

The following example swaps the environment tmp-dev with live-env.

```
$ eb swap
Select an environment to swap with.
1) staging-dev
2) live-env
(default is 1): 2
2018-07-11 21:05:25    INFO: swapEnvironmentCNAMEs is starting.
2018-07-11 21:05:26    INFO: Swapping CNAMEs for environments 'tmp-dev' and 'live-env'.
```

```
2018-07-11 21:05:30    INFO: 'tmp-dev.elasticbeanstalk.com' now points to 'awseb-e-j-
AWSEBLoa-M7U21VXNLWHN-487871449.us-west-2.elb.amazonaws.com'.
2018-07-11 21:05:30    INFO: Completed swapping CNAMEs for environments 'tmp-dev' and
 'live-env'.
```

The following example swaps the environment tmp-dev with the environment live-env but does not prompt you to enter or select a value for any settings.

```
$ eb swap tmp-dev --destination_name live-env
2018-07-11 21:18:12    INFO: swapEnvironmentCNAMEs is starting.
2018-07-11 21:18:13    INFO: Swapping CNAMEs for environments 'tmp-dev' and 'live-env'.
2018-07-11 21:18:17    INFO: 'tmp-dev.elasticbeanstalk.com' now points to 'awseb-e-j-
AWSEBLoa-M7U21VXNLWHN-487871449.us-west-2.elb.amazonaws.com'.
2018-07-11 21:18:17    INFO: Completed swapping CNAMEs for environments 'tmp-dev' and
 'live-env'.
```

eb tags

Description

Add, delete, update, and list tags of an Elastic Beanstalk resource.

For details about resource tagging in Elastic Beanstalk, see Tagging Elastic Beanstalk application resources.

Syntax

eb tags [*environment-name*] [--resource *ARN*] -l | --list

eb tags [*environment-name*] [--resource *ARN*] -a | --add *key1*=*value1*[,*key2*=*value2* ...]

eb tags [*environment-name*] [--resource *ARN*] -u | --update *key1*=*value1*[,*key2*=*value2* ...]

eb tags [*environment-name*] [--resource *ARN*] -d | --delete *key1*[,*key2* ...]

You can combine the `--add`, `--update`, and `--delete` subcommand options in a single command. At least one of them is required. You can't combined any of these three subcommand options with `--list`.

Without any additional arguments, all of these commands list or modify tags of the default environment in the current directory's application. With an *environment-name* argument, the commands list or modify tags of that environment. With the `--resource` option, the commands list or modify tags of any Elastic Beanstalk resource – an application, an environment, an application version, a saved configuration, or a custom platform version. Specify the resource by its Amazon Resource Name (ARN).

Options

None of these options are required. If you run **eb create** without any options, you are prompted to enter or select a value for each setting.

Name	Description

`-l`
 	
or
 	
`--list`
	List all tags that are currently applied to the resource.

`-a` *`key1`*`=`*`value1`*`[,`*`key2`*`=`*`value2`*
 	
or
 	
`--add` *`key1`*`=`*`value1`*`[,`*`key2`*`=`*`val`*
	Apply new tags to the resource. Specify tags as a comma-separated list of `key=value` pairs. You can't specify keys of existing tags.
	
	Valid values: See Tagging resources.

`-u` *`key1`*`=`*`value1`*`[,`*`key2`*`=`*`value2`*
 	
or
 	
`--updat`
`e` *`key1`*`=`*`value1`*`[,`*`key2`*`=`*`value2`* `.`
	Update the values of existing resource tags. Specify tags as a comma-separated list of `key=value` pairs. You must specify keys of existing tags.
	
	Valid values: See Tagging resources.

`-d` *`key1`*`[,`*`key2`* `...]`
 	
or
 	
`--delete` *`key1`*`[,`*`key2`* `...]`
	Delete existing resource tags. Specify tags as a comma-separated list of keys. You must specify keys of existing tags.
	
	Valid values: See Tagging resources.

`-r` *`region`*
 	
or
 	
`--region` *`region`*
	The AWS Region in which your resource exists.
	
	Default: the configured default region.

Name	Description
	For the list of values you can specify for this option, see AWS Elastic Beanstalk Endpoints and Quotas in the *AWS General Reference*.
`--resource` *ARN*	The ARN of the resource that the command modifies or lists tags for. If not specified, the command refers to the (default or specified) environment in the current directory's application. Valid values: See one of the sub-topic of Tagging resources that is specific to the resource you're intereste d in. These topics show how the resource's ARN is constructed and explain how to get a list of this resource's ARNs that exist for your application or account.

Output

The `--list` subcommand option displays a list of the resource's tags. The output shows both the tags that Elastic Beanstalk applies by default and your custom tags.

```
$ eb tags --list
Showing tags for environment 'MyApp-env':

Key                                 Value

Name                                MyApp-env
elasticbeanstalk:environment-id     e-63cmxwjaut
elasticbeanstalk:environment-name   MyApp-env
mytag                               tagvalue
tag2                                2nd value
```

The `--add`, `--update`, and `--delete` subcommand options, when successful, don't have any output. You can add the `--verbose` option to see detailed output of the command's activity.

```
$ eb tags --verbose --update "mytag=tag value"
Updated Tags:
```

```
Key                                 Value

mytag                               tag value
```

Examples

The following command successfully adds a tag with the key `tag1` and the value `value1` to the application's default environment, and at the same time deletes the tag `tag2`.

```
$ eb tags --add tag1=value1 --delete tag2
```

The following command successfully adds a tag to a saved configuration within an application.

```
$ eb tags --add tag1=value1 \
      --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-
id:configurationtemplate/my-app/my-template"
```

The following command fails because it tries to update a nonexisting tag.

```
$ eb tags --update tag3=newval
ERROR: Tags with the following keys can't be updated because they don't exist:

  tag3
```

The following command fails because it tries to update and delete the same key.

```
$ eb tags --update mytag=newval --delete mytag
ERROR: A tag with the key 'mytag' is specified for both '--delete' and '--update'. Each
 tag can be either deleted or updated in a single operation.
```

eb terminate

Description

Terminates the running environment so that you don't incur charges for unused AWS resources.

Using the `--all` option, deletes the application that the current directory was initialized to using **eb init**. The command terminates all environments in the application. It also terminates the application versions and saved configurations for the application, and then deletes the application.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command terminates the running custom environment.

> **Note**
>
> You can always launch a new environment using the same version later.

If you have data from an environment that you want to preserve, set the database deletion policy to `Retain` before terminating the environment. This keeps the database operational outside of Elastic Beanstalk. After this, any Elastic Beanstalk environments must connect to it as an external database. If you want to back up the data without keeping the database operational, set the deletion policy to take a snapshot of the database before terminating the environment. For more information, see Database lifecycle in the *Configuring environments* chapter of this guide.

> **Important**
>
> If you terminate an environment, you must also delete any CNAME mappings that you created, as other customers can reuse an available hostname. Be sure to delete DNS records that point to your terminated environment to prevent a *dangling DNS entry*. A dangling DNS entry can expose internet traffic destined for your domain to security vulnerabilities. It can also present other risks.
> For more information, see Protection from dangling delegation records in Route 53 in the *Amazon Route 53 Developer Guide*. You can also learn more about dangling DNS entries in Enhanced Domain Protections for Amazon CloudFront Requests in the *AWS Security Blog*.

Syntax

eb terminate

eb terminate *`environment-name`*

Options

Name	Description
`--all`	Terminates all environments in the application, the applicati on's application versions, and its saved configurations, and then deletes the application.
`--force`	Terminates the environment without prompting for confirmat ion.
`--ignore-links`	Terminates the environment even if there are dependent environments with links to it. See Compose Environments.
`--timeout`	The number of minutes before the command times out.

Output

If successful, the command returns the status of the `terminate` operation.

Example

The following example request terminates the environment tmp-dev.

```
$ eb terminate
The environment "tmp-dev" and all associated instances will be terminated.
To confirm, type the environment name: tmp-dev
2018-07-11 21:05:25    INFO: terminateEnvironment is starting.
2018-07-11 21:05:40    INFO: Deleted CloudWatch alarm named: awseb-e-2cpfjbra9a-stack-
AWSEBCloudwatchAlarmHigh-16V08YOF2KQ7U
2018-07-11 21:05:41    INFO: Deleted CloudWatch alarm named: awseb-e-2cpfjbra9a-stack-
AWSEBCloudwatchAlarmLow-6ZAWH9F20P7C
2018-07-11 21:06:42    INFO: Deleted Auto Scaling group policy named:
 arn:aws:autoscaling:us-east-2:11122223333:scalingPolicy:5d7d3e6b-
d59b-47c5-b102-3e11fe3047be:autoScalingGroupName/awseb-e-2cpfjbra9a-stack-
AWSEBAutoScalingGroup-7AXY7U13ZQ6E:policyName/awseb-e-2cpfjbra9a-stack-AWSEBAutoSca
lingScaleUpPolicy-1876U27JEC34J
2018-07-11 21:06:43    INFO: Deleted Auto Scaling group policy named:
 arn:aws:autoscaling:us-east-2:11122223333:scalingPolicy:29c6e7c7-7ac8-46fc-91f5-
```

```
cfabb65b985b:autoScalingGroupName/awseb-e-2cpfjbra9a-stack-
AWSEBAutoScalingGroup-7AXY7U13ZQ6E:policyName/awseb-e-2cpfjbra9a-stack-AWSEBAutoSca
lingScaleDownPolicy-SL4LHODMOMU
2018-07-11 21:06:48    INFO: Waiting for EC2 instances to terminate. This may take a
 few minutes.
2018-07-11 21:08:55    INFO: Deleted Auto Scaling group named: awseb-e-2cpfjbra9a-
stack-AWSEBAutoScalingGroup-7AXY7U13ZQ6E
2018-07-11 21:09:10    INFO: Deleted security group named: awseb-e-2cpfjbra9a-stack-
AWSEBSecurityGroup-XT4YYGFL7I99
2018-07-11 21:09:40    INFO: Deleted load balancer named: awseb-e-2-AWSEBLoa-
AK6RRYFQVV3S
2018-07-11 21:09:42    INFO: Deleting SNS topic for environment tmp-dev.
2018-07-11 21:09:52    INFO: terminateEnvironment completed successfully.
```

eb upgrade

Description

Upgrades the platform of your environment to the most recent version of the platform on which it is currently running.

If the root directory contains a `platform.yaml` file specifying a custom platform, this command upgrades the environment to the most recent version of the custom platform on which it is currently running.

Syntax

eb upgrade

eb upgrade ***`environment-name`***

Options

Name	Description
`--force`	Upgrades without requiring you to confirm the environment name before starting the upgrade process.
`--noroll`	Updates all instances without using rolling updates to keep some instances in service during the upgrade.

Name	Description
Common options	

Output

The command shows an overview of the change and prompts you to confirm the upgrade by typing the environment name. If successful, your environment is updated and then launched with the most recent version of the platform.

Example

The following example upgrades the current platform version of the specified environment to the most recently available platform version.

```
$ eb upgrade
Current platform: 64bit Amazon Linux 2014.09 v1.0.9 running Python 2.7
Latest platform:  64bit Amazon Linux 2014.09 v1.2.0 running Python 2.7

WARNING: This operation replaces your instances with minimal or zero downtime. You may
 cancel the upgrade after it has started by typing "eb abort".
You can also change your platform version by typing "eb clone" and then "eb swap".

To continue, type the environment name:
```

eb use

Description

Sets the specified environment as the default environment.

When using Git, **eb use** sets the default environment for the current branch. Run this command once in each branch that you want to deploy to Elastic Beanstalk.

Syntax

eb use ***environment-name***

Options

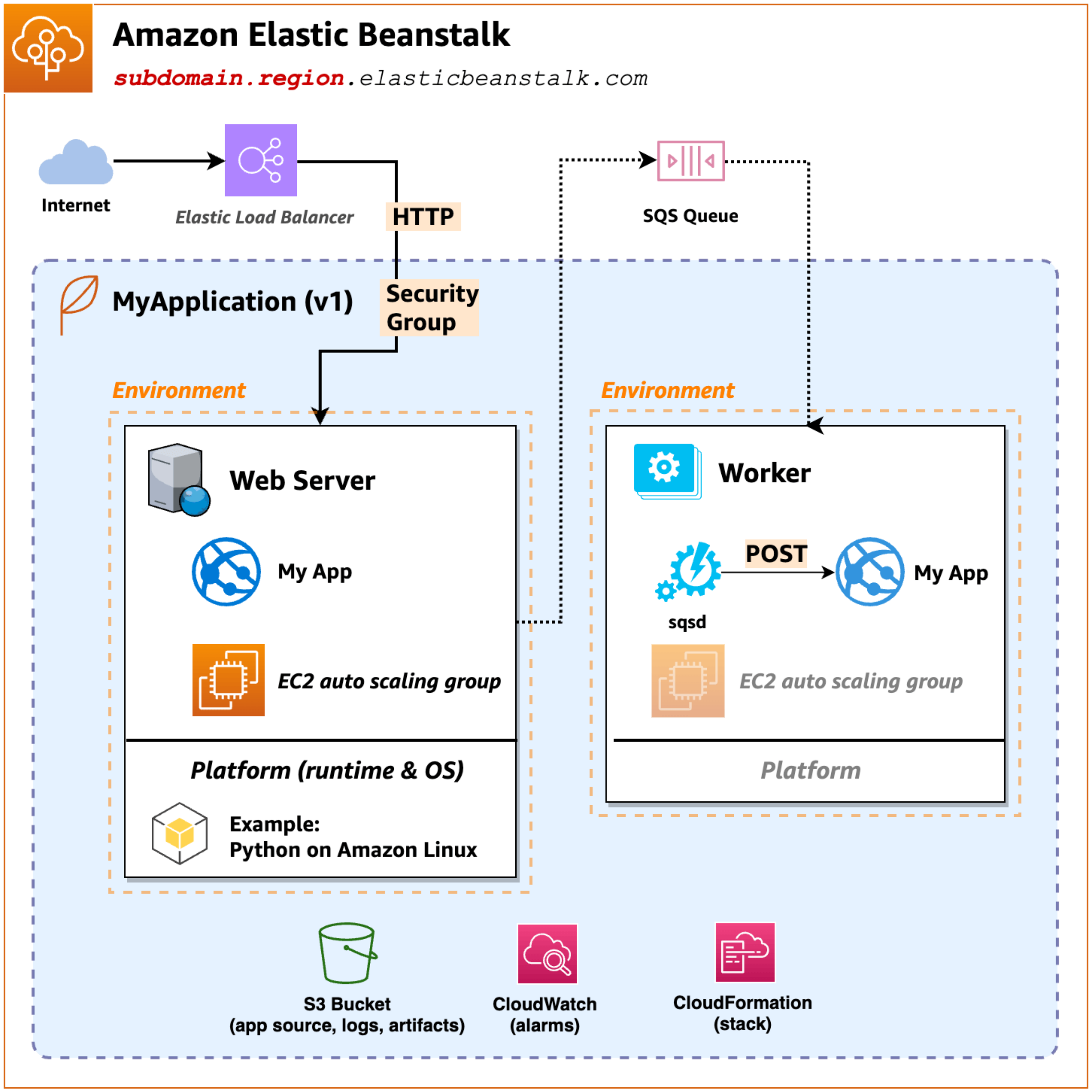
Name	Description
`--source codecommit/ repository-name/branch-name`	CodeCommit repository and branch.

`-r region`
 	
`--region region`
	Change the region in which you create environments.

Common options	

Understanding concepts in Elastic Beanstalk

Becoming familiar with the concepts and terms will help you gain an understanding needed for deploying your applications with Elastic Beanstalk.



Application

An Elastic Beanstalk *application* is a container for Elastic Beanstalk components, including *environments*, *versions*, and *environment configurations*. Within an Elastic Beanstalk application, you manage all the resources relevant to running your code.

Application version

In Elastic Beanstalk, an *application version* refers to a specific, labeled iteration of deployable code for a web application. An application version points to an Amazon Simple Storage Service (Amazon S3) object that contains the deployable code, such as a Java WAR file.

An application version is part of an application. Applications can have many versions and each application version is unique. In a running environment, you can deploy any application version you already uploaded to the application, or you can upload and immediately deploy a new application version. For example, you could upload multiple application versions to test differences between them.

Environment

An *environment* is a collection of AWS resources running an application version. Each environment runs only one application version at a time, however, you can run the same application version or different application versions in many environments simultaneously. When you create an environment, Elastic Beanstalk provisions the resources needed in your AWS account to run the application version you specified.

Environment tier

When you launch an Elastic Beanstalk environment, you first choose an environment tier. The environment tier designates the type of application that the environment runs and determines what resources Elastic Beanstalk provisions to support it. An application that serves HTTP requests runs in a web server environment tier. A backend environment that pulls tasks from an Amazon Simple Queue Service (Amazon SQS) queue runs in a worker environment tier.

Environment configuration

An *environment configuration* identifies a collection of parameters and settings that define how an environment and its associated resources behave. When you update an environment's configuration settings, Elastic Beanstalk automatically applies the changes to existing resources or deletes and deploys new resources (depending on the type of change).

Saved configuration

A *saved configuration* is a template that you can use as a starting point for creating unique environment configurations. You can create and modify saved configurations, and apply them to environments, using the Elastic Beanstalk console, EB CLI, AWS CLI, or API. The API and the AWS CLI refer to saved configurations as *configuration templates*.

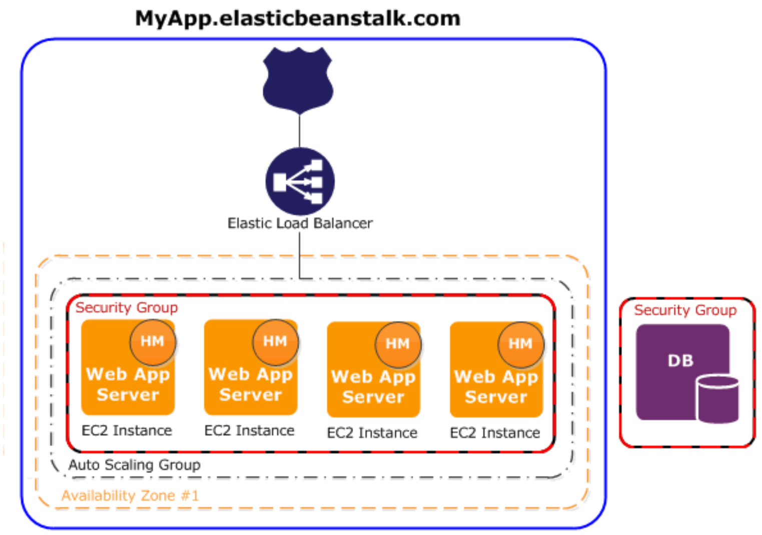
Platform

A *platform* is a combination of an operating system, programming language runtime, web server, application server, and Elastic Beanstalk components. You design and target your web application to a platform. Elastic Beanstalk provides a variety of platforms on which you can build your applications.

For details, see Elastic Beanstalk platforms.

Elastic Beanstalk web server environments

The following diagram shows an example Elastic Beanstalk architecture for a web server environment tier, and shows how the components in that type of environment tier work together.



The environment is the heart of the application. In the diagram, the environment is shown within the top-level solid line. When you create an environment, Elastic Beanstalk provisions the resources required to run your application. AWS resources created for an environment include one elastic load balancer (ELB in the diagram), an Auto Scaling group, and one or more Amazon Elastic Compute Cloud (Amazon EC2) instances.

Every environment has a CNAME (URL) that points to a load balancer. The environment has a URL, such as `myapp.us-west-2.elasticbeanstalk.com`. This URL is aliased in Amazon Route 53 to an Elastic Load Balancing URL—something like `abcdef-123456.us-west-2.elb.amazonaws.com`—by using a CNAME record. Amazon Route 53 is a highly available and scalable Domain Name System (DNS) web service. It provides secure and reliable routing to your infrastructure. Your domain name that you registered with your DNS provider will forward requests to the CNAME.

The load balancer sits in front of the Amazon EC2 instances, which are part of an Auto Scaling group. Amazon EC2 Auto Scaling automatically starts additional Amazon EC2 instances to accommodate increasing load on your application. If the load on your application decreases, Amazon EC2 Auto Scaling stops instances, but always leaves at least one instance running.

The software stack running on the Amazon EC2 instances is dependent on the *container type*. A container type defines the infrastructure topology and software stack to be used for that environment. For example, an Elastic Beanstalk environment with an Apache Tomcat container uses the Amazon Linux operating system, Apache web server, and Apache Tomcat software. For a list of supported container types, see Elastic Beanstalk supported platforms. Each Amazon EC2 instance that runs your application uses one of these container types. In addition, a software component called the *host manager (HM)* runs on each Amazon EC2 instance. The host manager is responsible for the following:

- Deploying the application
- Aggregating events and metrics for retrieval via the console, the API, or the command line
- Generating instance-level events
- Monitoring the application log files for critical errors
- Monitoring the application server
- Patching instance components
- Rotating your application's log files and publishing them to Amazon S3

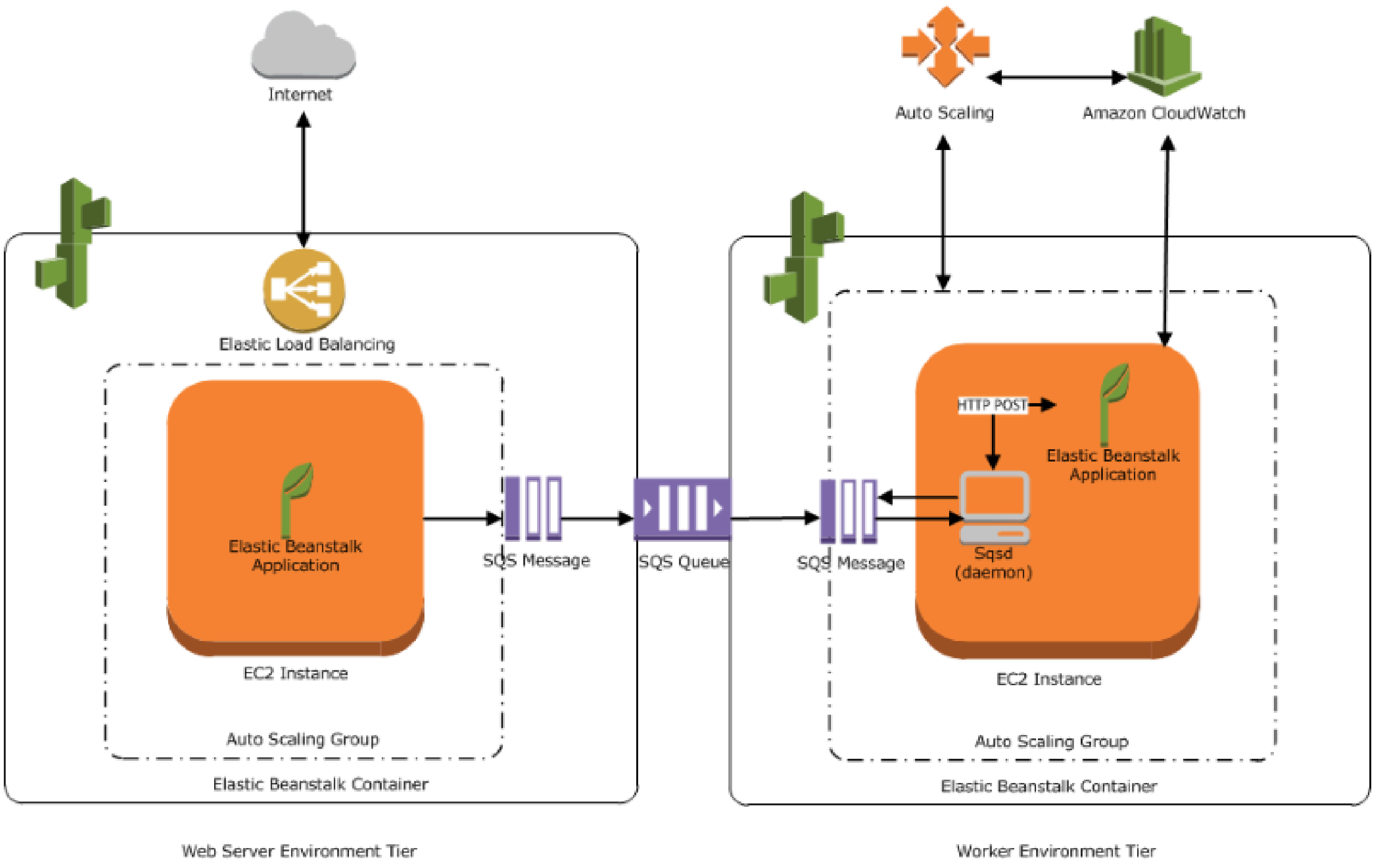
The host manager reports metrics, errors and events, and server instance status, which are available via the Elastic Beanstalk console, APIs, and CLIs.

The Amazon EC2 instances shown in the diagram are part of one *security group*. A security group defines the firewall rules for your instances. By default, Elastic Beanstalk defines a security group, which allows everyone to connect using port 80 (HTTP). You can define more than one security group. For example, you can define a security group for your database server. For more information about Amazon EC2 security groups and how to configure them for your Elastic Beanstalk application, see EC2 security groups.

Elastic Beanstalk worker environments

AWS resources created for a worker environment tier include an Auto Scaling group, one or more Amazon EC2 instances, and an IAM role. For the worker environment tier, Elastic Beanstalk also creates and provisions an Amazon SQS queue if you don't already have one. When you launch a worker environment, Elastic Beanstalk installs the necessary support files for your programming language of choice and a daemon on each EC2 instance in the Auto Scaling group. The daemon reads messages from an Amazon SQS queue. The daemon sends data from each message that it reads to the web application running in the worker environment for processing. If you have multiple instances in your worker environment, each instance has its own daemon, but they all read from the same Amazon SQS queue.

The following diagram shows the different components and their interactions across environments and AWS services.



Amazon CloudWatch is used for alarms and health monitoring. For more information, go to Basic health reporting.

For details about how the worker environment tier works, see Elastic Beanstalk worker environments.

Design considerations for your Elastic Beanstalk applications

Because applications deployed using AWS Elastic Beanstalk run on AWS Cloud resources, you should keep several configuration factors in mind to optimize your applications: *scalability*, *security*, *persistent storage*, *fault tolerance*, *content delivery*, *software updates and patching*, and *connectivity*. Each of these are covered separately in this topic. For a comprehensive list of technical AWS whitepapers, covering topics such as architecture, as well as security and economics, see AWS Cloud Computing Whitepapers.

Scalability

When operating in a physical hardware environment, in contrast to a cloud environment, you can approach scalability in one of either two ways. Either you can scale up through vertical scaling or you can scale out through horizontal scaling. The scale-up approach requires that you invest in powerful hardware, which can support the increasing demands of your business. The scale-out approach requires that you follow a distributed model of investment. As such, your hardware and application acquisitions can be more targeted, your data sets are federated, and your design is service oriented. The scale-up approach can be expensive, and there's also the risk that your demand could outgrow your capacity. In this regard, the scale-out approach is usually more effective. However, when using it, you must be able to predict demand at regular intervals and deploy infrastructure in chunks to meet that demand. As a result, this approach can often lead to unused capacity and might require some careful monitoring.

By migrating to the cloud, you can make your infrastructure align well with demand by leveraging the elasticity of cloud. Elasticity helps to streamline resource acquisition and release. With it, your infrastructure can rapidly scale in and scale out as demand fluctuates. To use it, configure your Auto Scaling settings to scale up or down based on the metrics for the resources in your environment. For example, you can set metrics such as server utilization or network I/O. You can use Auto Scaling for compute capacity to be added automatically whenever usage rises and for it to be removed whenever usage drops. You can publish system metrics (for example, CPU, memory, disk I/O, and network I/O) to Amazon CloudWatch. Then, you can use CloudWatch to configure alarms to trigger Auto Scaling actions or send notifications based on these metrics. For instructions on how to configure Auto Scaling, see Auto Scaling your Elastic Beanstalk environment instances.

We also recommend that you design all your Elastic Beanstalk applications as *stateless* as possible, using loosely coupled, fault-tolerant components that can be scaled out as needed. For more information about designing scalable application architectures for AWS, see *AWS Well-Architected Framework*.

Security

Security on AWS is a shared responsibility. Amazon Web Services protects the physical resources in your environment and ensures that the Cloud is a safe place for you to run applications. You're responsible for the security of data coming in and out of your Elastic Beanstalk environment and the security of your application.

Configure SSL to protect information that flows between your application and clients. To configure SSL, you need a free certificate from AWS Certificate Manager (ACM). If you already have a certificate from an external certificate authority (CA), you can use ACM to import that your certificate. Otherwise, you can import it using the AWS CLI.

If ACM isn't available in your AWS Region, you can purchase a certificate from an external CA, such as VeriSign or Entrust. Then, use the AWS Command Line Interface (AWS CLI) to upload a third-party or self-signed certificate and private key to AWS Identity and Access Management (IAM). The public key of the certificate authenticates your server to the browser. It also serves as the basis for creating the shared session key that encrypts the data in both directions. For instructions on how to create, upload, and assign an SSL certificate to your environment, see Configuring HTTPS for your Elastic Beanstalk environment.

When you configure an SSL certificate for your environment, data is encrypted between the client and the Elastic Load Balancing load balancer for your environment. By default, encryption is terminated at the load balancer, and traffic between the load balancer and Amazon EC2 instances is unencrypted.

Persistent storage

Elastic Beanstalk applications run on Amazon EC2 instances that have no persistent local storage. When the Amazon EC2 instances terminate, the local file system isn't saved. New Amazon EC2 instances start with a default file system. We recommend that you configure your application to store data in a persistent data source. AWS offers a number of persistent storage services that you can use for your application. The following table lists them.

Storage service	Service documentation	Elastic Beanstalk integration
Amazon S3	Amazon Simple Storage Service Documentation	Using Elastic Beanstalk with Amazon S3
Amazon Elastic File System	Amazon Elastic File System Documentation	Using Elastic Beanstalk with Amazon Elastic File System
Amazon Elastic Block Store	Amazon Elastic Block Store Feature Guide: Elastic Block Store	
Amazon DynamoDB	Amazon DynamoDB Documentation	Using Elastic Beanstalk with Amazon DynamoDB

Storage service	Service documentation	Elastic Beanstalk integration
Amazon Relational Database Service (RDS)	Amazon Relational Database Service Documentation	Using Elastic Beanstalk with Amazon RDS

Note

Elastic Beanstalk creates a *webapp* user for you to set up as the owner of application directories on EC2 instances. For Amazon Linux 2 platform versions that are released on or after Feburary 3, 2022, Elastic Beanstalk assigns the *webapp* user a uid (user id) and gid (group id) value of 900 for new environments. It does the same for existing environments following a platform version update. This approach keeps consistent access permission for the *webapp* user to permanent file system storage.
In the unlikely situation that another user or process is already using 900, the operating system defaults the *webapp* user uid and gid to another value. Run the Linux command **id webapp** on your EC2 instances to verify the uid and gid values that are assigned to the *webapp* user.

Fault tolerance

As a rule of thumb, you should be a pessimist when designing architecture for the cloud. Leverage the elasticity that it offers. Always design, implement, and deploy for automated recovery from failure. Use multiple Availability Zones for your Amazon EC2 instances and for Amazon RDS. Availability Zones are conceptually like logical data centers. Use Amazon CloudWatch to get more visibility into the health of your Elastic Beanstalk application and take appropriate actions in case of hardware failure or performance degradation. Configure your Auto Scaling settings to maintain your fleet of Amazon EC2 instances at a fixed size so that unhealthy Amazon EC2 instances are replaced by new ones. If you're using Amazon RDS, then set the retention period for backups, so that Amazon RDS can perform automated backups.

Content delivery

When users connect to your website, their requests may be routed through a number of individual networks. As a result, users might experience poor performance due to high latency. Amazon CloudFront can help ameliorate latency issues by distributing your web content, such as images and video, across a network of edge locations around the world. Users' requests are routed to the nearest edge location, so content is delivered with the best possible performance. CloudFront works seamlessly with Amazon S3, which durably stores the original, definitive versions of your files. For more information about Amazon CloudFront, see the Amazon CloudFront Developer Guide.

Software updates and patching

AWS Elastic Beanstalk regularly releases platform updates to provide fixes, software updates, and new features. Elastic Beanstalk offers several options to handle platform updates. With managed platform updates your environment automatically upgrades to the latest version of a platform during a scheduled maintenance window while your application remains in service. For environments created on November 25, 2019 or later using the Elastic Beanstalk console, managed updates are enabled by default whenever possible. You can also manually initiate updates using the Elastic Beanstalk console or EB CLI.

Connectivity

Elastic Beanstalk needs to be able to connect to the instances in your environment to complete deployments. When you deploy an Elastic Beanstalk application inside an Amazon VPC, the configuration required to enable connectivity depends on the type of Amazon VPC environment you create:

- For single-instance environments, no additional configuration is required. This is because, with these environments, Elastic Beanstalk assigns each Amazon EC2 instance a public Elastic IP address that enables the instance to communicate directly with the internet.
- For load-balanced, scalable environments in an Amazon VPC with both public and private subnets, you must do the following:
  - Create a load balancer in the public subnet to route inbound traffic from the internet to the Amazon EC2 instances.
  - Create a network address translation (NAT) device to route outbound traffic from the Amazon EC2 instances in private subnets to the internet.
  - Create inbound and outbound routing rules for the Amazon EC2 instances inside the private subnet.
  - If you're using a NAT instance, configure the security groups for the NAT instance and Amazon EC2 instances to enable internet communication.
- For a load-balanced, scalable environment in an Amazon VPC that has one public subnet, no additional configuration is required. This is because, with this environment, your Amazon EC2

instances are configured with a public IP address that enables the instances to communicate with the internet.

For more information about using Elastic Beanstalk with Amazon VPC, see Using Elastic Beanstalk with Amazon VPC.

Managing Elastic Beanstalk applications

This chapter describes how to manage and configure your Elastic Beanstalk applications. The first step in using AWS Elastic Beanstalk is to create an application, which represents your web application in AWS. In Elastic Beanstalk an application serves as a container for the environments that run your web app and for versions of your web app's source code, saved configurations, logs, and other artifacts that you create while using Elastic Beanstalk.

To create an application

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose **Create application**.
3. Use the on-screen form to provide an application name.
4. (Optional) Provide a description, and add tag keys and values.
5. Choose **Create**.

After creating the application, the console prompts you to create an environment for it. For detailed information about all of the options available, see Creating an Elastic Beanstalk environment.

If you no longer need an application, you can delete it.

Warning

Deleting an application terminates all associated environments and deletes all application versions and saved configurations that belong to the application.

To delete an application

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then select your application on the list.
3. Choose **Actions**, and then choose **Delete application**.

Topics

- Elastic Beanstalk application management console
- Managing application versions
- Create an Elastic Beanstalk application source bundle
- Using the EB CLI with AWS CodeBuild
- Tagging applications
- Tagging Elastic Beanstalk application resources

Elastic Beanstalk application management console

This topic explains how you can use the AWS Elastic Beanstalk console to manage applications, application versions, and saved configurations.

To access the application management console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.

   The application overview page shows a list with an overview of all environments associated with the application.
3. You have a few ways to continue:

   a. From the **Actions** drop-down menu, you can choose one of the application management actions: **Create environment**, **Delete application**, **View application versions**, **View saved configurations**, **Restore terminated environment**.

      To launch an environment in this application, you can directly choose **Create environment**. For details, see the section called "Creating environments".

   b. The page lists the environment name next to applications that are deployed to an environment. Choose an environment name to go to the environment management console for that environment, where you can configure, monitor, or manage the environment.

   c. When you select an application from the list, the left navigation pane lists the application.

      - Choose **Application versions** following the application name in the navigation pane to view and manage the application versions for your application.

An application version is an uploaded version of your application code. You can upload new versions, deploy an existing version to any of the application's environments, or delete old versions. For more information, see Managing application versions.

- Choose **Saved configurations** following the application name in the navigation pane to view and manage configurations saved from running environments.

  A saved configuration is a collection of settings that you can use to restore an environment's settings to a previous state, or to create an environment with the same settings. For more information see Using Elastic Beanstalk saved configurations.

Managing application versions

This topic explains application versions and how to create and manage them.

Elastic Beanstalk creates an application version whenever you upload source code. This usually occurs when you create an environment or upload and deploy code using the environment management console or EB CLI. Elastic Beanstalk deletes these application versions according to the application's lifecycle policy and when you delete the application. For details about application lifecycle policy, see Configuring application version lifecycle settings.

You can also upload a source bundle without deploying it from the application management console or with the EB CLI command **eb appversion**. Elastic Beanstalk stores source bundles in Amazon Simple Storage Service (Amazon S3) and doesn't automatically delete them.

You can apply tags to an application version when you create it, and edit tags of existing application versions. For details, see Tagging application versions.

Creating application versions

You can also create a new application version using the EB CLI. For more information, see **eb appversion** in the *EB CLI commands* chapter.

Note

Over time, your application can accumulate many application versions. To save storage space and avoid hitting the application version quota, it's a good idea to delete application versions that you no longer need.

The file you specify in the following procedure is associated with your application. You can deploy the application version to a new or existing environment.

To create a new application version

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.
3. In the navigation pane, find your application's name and choose **Application versions**.
4. Choose **Upload**. Use the on-screen form to upload your application's source bundle.

   Note

   The source bundle's file size limit is 500 MB.
5. Optionally, provide a brief description, and add tag keys and values.
6. Choose **Upload**.

Deleting application versions

You can also delete an application version using the EB CLI. For more information, see **eb appversion** in the *EB CLI commands* chapter.

Note

Deleting an application version doesn't affect environments currently running that version.

You can also configure Elastic Beanstalk to delete old versions automatically by configuring application version lifecycle settings. If you configure these lifecycle settings, they're applied when you create new application versions. For example, if you configure a maximum of 25 application versions, Elastic Beanstalk deletes the oldest version when you upload a 26th version. If you set a maximum age of 90 days, any versions older than 90 days are deleted when you upload a new version. For details, see the section called "Version lifecycle".

To delete an application version

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.
3. In the navigation pane, find your application's name and choose **Application versions**.
4. Select one or more application versions that you want to delete.
5. Choose **Actions**, then choose **Delete**.
6. (Optional) To leave the application source bundle for these application versions in your Amazon Simple Storage Service (Amazon S3) bucket, clear the box for **Delete versions from Amazon S3**.
7. Choose **Delete**.

If you don't choose to delete the source bundle from Amazon S3, Elastic Beanstalk still deletes the version from its records. However, the source bundle is left in your Elastic Beanstalk storage bucket. The application version quota applies only to versions Elastic Beanstalk tracks. Therefore, you can delete versions to stay within the quota, but retain all source bundles in Amazon S3.

Note

The application version quota doesn't apply to source bundles, but you might still incur Amazon S3 charges, and retain personal information beyond the time you need it. Elastic Beanstalk never deletes source bundles automatically. You should delete source bundles when you no longer need them.

Configuring application version lifecycle settings

This topic explains the policies and quotas that Elastic Beanstalk applies to the versions of your application in a given environment, including how long an application version remains in an environment.

Each time you upload a new version of your application with the Elastic Beanstalk console or the EB CLI, Elastic Beanstalk creates an application version. If you don't delete versions that you no longer use, you will eventually reach the application version quota and be unable to create new versions of that application.

You can avoid hitting the quota by applying an *application version lifecycle policy* to your applications. A lifecycle policy tells Elastic Beanstalk to delete application versions that are old, or to delete application versions when the total number of versions for an application exceeds a specified number.

Elastic Beanstalk applies an application's lifecycle policy each time you create a new application version, and deletes up to 100 versions each time the lifecycle policy is applied. Elastic Beanstalk deletes old versions after creating the new version, and does not count the new version towards the maximum number of versions defined in the policy.

Elastic Beanstalk does not delete application versions that are currently being used by an environment, or application versions deployed to environments that were terminated less than ten weeks before the policy was triggered.

The application version quota applies across all applications in a region. If you have several applications, configure each application with a lifecycle policy appropriate to avoid reaching the quota. For example, if you have 10 applications in a region and the quota is 1,000 application versions, consider setting a lifecycle policy with a quota of 99 application versions for all applications, or set other values in each application as long as the total is less than 1,000 application versions. Elastic Beanstalk only applies the policy if the application version creation succeeds, so if you have already reached the quota, you must delete some versions manually prior to creating a new version.

By default, Elastic Beanstalk leaves the application version's source bundle in Amazon S3 to prevent loss of data. You can delete the source bundle to save space.

You can set the lifecycle settings through the Elastic Beanstalk CLI and APIs. See **eb appversion**, CreateApplication (using the `ResourceLifecycleConfig` parameter), and UpdateApplicationResourceLifecycle for details.

Setting the application lifecycle settings in the console

You can specify the lifecycle settings in the Elastic Beanstalk console.

To specify your application lifecycle settings

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.
3. In the navigation pane, find your application's name and choose **Application versions**.
4. Choose **Settings**.
5. Use the on-screen form to configure application lifecycle settings.
6. Choose **Save**.

On the settings page, you can do the following.

- Configure lifecycle settings based on the total count of application versions or the age of application versions.
- Specify whether to delete the source bundle from S3 when the application version is deleted.
- Specify the service role under which the application version is deleted. To include all permissions required for version deletion, choose the default Elastic Beanstalk service role, named `aws-elasticbeanstalk-service-role`, or another service role using the Elastic Beanstalk managed service policies. For more information, see Managing Elastic Beanstalk service roles.

Tagging application versions

This topic explains the benefits of tagging your Elastic Beanstalk application versions and how to manage the tags.

You can apply tags to your AWS Elastic Beanstalk application versions. Tags are key-value pairs associated with AWS resources. For information about Elastic Beanstalk resource tagging, use cases, tag key and value constraints, and supported resource types, see Tagging Elastic Beanstalk application resources.

You can specify tags when you create an application version. In an existing application version, you can add or remove tags, and update the values of existing tags. You can add up to 50 tags to each application version.

Adding tags during application version creation

When you use the Elastic Beanstalk console to create an environment, and you choose to upload a version of your application code, you can specify tag keys and values to associate with the new application version.

You can also use the Elastic Beanstalk console to upload an application version without immediately using it in an environment. You can specify tag keys and values when you upload an application version.

With the AWS CLI or other API-based clients, add tags by using the `--tags` parameter on the **create-application-version** command.

```
$ aws elasticbeanstalk create-application-version \
      --tags Key=mytag1,Value=value1 Key=mytag2,Value=value2 \
      --application-name my-app --version-label v1
```

When you use the EB CLI to create or update an environment, an application version is created from the code that you deploy. There isn't a direct way to tag an application version during its creation through the EB CLI. See the following section to learn about adding tags to an existing application version.

Managing tags of an existing application version

You can add, update, and delete tags in an existing Elastic Beanstalk application version.

To manage an application version's tags using the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.
3. In the navigation pane, find your application's name and choose **Application versions**.
4. Select the application version you want to manage.
5. Choose **Actions**, and then choose **Manage tags**.
6. Use the on-screen form to add, update, or delete tags.
7. To save the changes choose **Apply** at the bottom of the page.

If you use the EB CLI to update your application version, use **eb tags** to add, update, delete, or list tags.

For example, the following command lists the tags in an application version.

```
~/workspace/my-app$ eb tags --list --resource "arn:aws:elasticbeanstalk:us-east-2:my-
account-id:applicationversion/my-app/my-version"
```

The following command updates the tag `mytag1` and deletes the tag `mytag2`.

```
~/workspace/my-app$ eb tags --update mytag1=newvalue --delete mytag2 \
      --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-
id:applicationversion/my-app/my-version"
```

For a complete list of options and more examples, see `eb tags`.

With the AWS CLI or other API-based clients, use the **list-tags-for-resource** command to list the tags of an application version.

```
$ aws elasticbeanstalk list-tags-for-resource --resource-arn
 "arn:aws:elasticbeanstalk:us-east-2:my-account-id:applicationversion/my-app/my-
version"
```

Use the **update-tags-for-resource** command to add, update, or delete tags in an application version.

```
$ aws elasticbeanstalk update-tags-for-resource \
      --tags-to-add Key=mytag1,Value=newvalue --tags-to-remove mytag2 \
      --resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-
id:applicationversion/my-app/my-version"
```

Specify both tags to add and tags to update in the `--tags-to-add` parameter of **update-tags-for-resource**. A nonexisting tag is added, and an existing tag's value is updated.

Note

To use some of the EB CLI and AWS CLI commands with an Elastic Beanstalk application version, you need the application version's ARN. You can retrieve the ARN by using the following command.

```
$ aws elasticbeanstalk describe-application-versions --application-name my-app
 --version-label my-version
```

Create an Elastic Beanstalk application source bundle

This topic explains how to upload your application source files to Elastic Beanstalk in a source bundle. It explains the requirements of a source bundle, the structure, and the approaches to create one.

When you use the AWS Elastic Beanstalk console to deploy a new application or an application version, you'll need to upload the files for the application in a *source bundle*. Your source bundle must meet the following requirements:

- Consist of a single `ZIP` file or `WAR` file (you can include multiple `WAR` files inside your `ZIP` file)
- Not exceed 500 MB
- Not include a parent folder or top-level directory (subdirectories are fine)

If you want to deploy a worker application that processes periodic background tasks, your application source bundle must also include a `cron.yaml` file. For more information, see Periodic tasks.

If you are deploying your application with the Elastic Beanstalk Command Line Interface (EB CLI), the AWS Toolkit for Eclipse, or the AWS Toolkit for Visual Studio, the ZIP or WAR file will automatically be structured correctly. For more information, see Setting up the EB command line interface (EB CLI) to manage Elastic Beanstalk, Deploying Java applications with Elastic Beanstalk, and The AWS Toolkit for Visual Studio.

Sections

Creating a source bundle from the command line

Create a source bundle using the `zip` command. To include hidden files and folders, use a pattern like the following.

```
~/myapp$ zip ../myapp.zip -r * .[^.]*
  adding: app.js (deflated 63%)
  adding: index.js (deflated 44%)
  adding: manual.js (deflated 64%)
  adding: package.json (deflated 40%)
  adding: restify.js (deflated 85%)
  adding: .ebextensions/ (stored 0%)
  adding: .ebextensions/xray.config (stored 0%)
```

This ensures that Elastic Beanstalk configuration files and other files and folders that start with a period are included in the archive.

For Tomcat web applications, use `jar` to create a web archive.

```
~/myapp$ jar -cvf myapp.war .
```

The above commands include hidden files that may increase your source bundle size unnecessarily. For more control, use a more detailed file pattern, or create your source bundle with Git.

Creating a source bundle with Git

If you're using Git to manage your application source code, use the `git archive` command to create your source bundle.

```
$ git archive -v -o myapp.zip --format=zip HEAD
```

`git archive` only includes files that are stored in git, and excludes ignored files and git files. This helps keep your source bundle as small as possible. For more information, go to the git-archive manual page.

Zipping files in Mac OS X Finder or Windows explorer

When you create a `ZIP` file in Mac OS X Finder or Windows Explorer, make sure you zip the files and subfolders themselves, rather than zipping the parent folder.

Note

The graphical user interface (GUI) on Mac OS X and Linux-based operating systems does not display files and folders with names that begin with a period (.). Use the command line instead of the GUI to compress your application if the ZIP file must include a hidden folder, such as .ebextensions. For command line procedures to create a ZIP file on Mac OS X or a Linux-based operating system, see Creating a source bundle from the command line.

Example

Suppose you have a Python project folder labeled myapp, which includes the following files and subfolders:

```
myapplication.py
README.md
static/
static/css
static/css/styles.css
static/img
static/img/favicon.ico
static/img/logo.png
templates/
templates/base.html
templates/index.html
```

As noted in the list of requirements above, your source bundle must be compressed without a parent folder, so that its decompressed structure does not include an extra top-level directory. In this example, no myapp folder should be created when the files are decompressed (or, at the command line, no myapp segment should be added to the file paths).

This sample file structure is used throughout this topic to illustrate how to zip files.

Creating a source bundle for a .NET application

If you use Visual Studio, you can use the deployment tool included in the AWS Toolkit for Visual Studio to deploy your .NET application to Elastic Beanstalk. For more information, see Deploying Elastic Beanstalk applications in .NET using the deployment tool.

If you need to manually create a source bundle for your .NET application, you cannot simply create a `ZIP` file that contains the project directory. You must create a web deployment package for your project that is suitable for deployment to Elastic Beanstalk. There are several methods you can use to create a deployment package:

- Create the deployment package using the **Publish Web** wizard in Visual Studio. For more information, go to How to: Create a Web Deployment Package in Visual Studio.

  Important

  When creating the web deployment package, you must start the **Site name** with `Default Web Site`.

- If you have a .NET project, you can create the deployment package using the **msbuild** command as shown in the following example.

  Important

  The `DeployIisAppPath` parameter must begin with `Default Web Site`.

```
C:/> msbuild <web_app>.csproj /t:Package /p:DeployIisAppPath="Default Web Site"
```

- If you have a website project, you can use the IIS Web Deploy tool to create the deployment package. For more information, go to Packaging and Restoring a Web site.

  Important

  The `apphostconfig` parameter must begin with `Default Web Site`.

If you are deploying multiple applications or an ASP.NET Core application, put your `.ebextensions` folder in the root of the source bundle, side by side with the application bundles and manifest file:

```
~/workspace/source-bundle/
|-- .ebextensions
|   |-- environmentvariables.config
|   `-- healthcheckurl.config
```

```
|-- AspNetCore101HelloWorld.zip
|-- AspNetCoreHelloWorld.zip
|-- aws-windows-deployment-manifest.json
`-- VS2015AspNetWebApiApp.zip
```

Testing your source bundle

You may want to test your source bundle locally before you upload it to Elastic Beanstalk. Because Elastic Beanstalk essentially uses the command line to extract the files, it's best to do your tests from the command line rather than with a GUI tool.

Ensure that the decompressed files appear in the same folder as the archive itself, rather than in a new top-level folder or directory.

Using the EB CLI with AWS CodeBuild

AWS CodeBuild compiles your source code, runs unit tests, and produces artifacts that are ready to deploy. You can use CodeBuild together with the EB CLI to automate building your application from its source code. Environment creation and each deployment thereafter start with a build step, and then deploy the resulting application.

Note

Some regions don't offer CodeBuild. The integration between Elastic Beanstalk and CodeBuild doesn't work in these regions.
For information about the AWS services offered in each region, see Region Table.

Creating an application

To create an Elastic Beanstalk application that uses CodeBuild

1. Include a CodeBuild build specification file, `buildspec.yml`, in your application folder.
2. Add an `eb_codebuild_settings` entry with options specific to Elastic Beanstalk to the file.
3. Run **eb init** in the folder.

Note

Do not use the period (`.`) or space (` `) characters in *Application name* when you use the EB CLI with CodeBuild.

Elastic Beanstalk extends the CodeBuild build specification file format to include the following additional settings:

```
eb_codebuild_settings:
  CodeBuildServiceRole: role-name
  ComputeType: size
  Image: image
  Timeout: minutes
```

`CodeBuildServiceRole`

The ARN or name of the AWS Identity and Access Management (IAM) service role that CodeBuild can use to interact with dependent AWS services on your behalf. This value is required. If you omit it, any subsequent **eb create** or **eb deploy** command fails.

To learn more about creating a service role for CodeBuild, see Create a CodeBuild Service Role in the *AWS CodeBuild User Guide*.

Note

You also need permissions to perform actions in CodeBuild itself. The Elastic Beanstalk **AdministratorAccess-AWSElasticBeanstalk** managed user policy includes all the required CodeBuild action permissions. If you're not using the managed policy, be sure to allow the following permissions in your user policy.

```
  "codebuild:CreateProject",
  "codebuild:DeleteProject",
  "codebuild:BatchGetBuilds",
  "codebuild:StartBuild"
```

For details, see Managing Elastic Beanstalk user policies.

`ComputeType`

The amount of resources used by the Docker container in the CodeBuild build environment. Valid values are BUILD_GENERAL1_SMALL, BUILD_GENERAL1_MEDIUM, and BUILD_GENERAL1_LARGE.

`Image`

The name of the Docker Hub or Amazon ECR image that CodeBuild uses for the build environment. This Docker image should contain all the tools and runtime libraries required to build your code, and should match your application's target platform. CodeBuild manages and maintains a set of images specifically meant to be used with Elastic Beanstalk. It is recommended that you use one of them. For details, see Docker Images Provided by CodeBuild in the *AWS CodeBuild User Guide*.

The `Image` value is optional. If you omit it, the **eb init** command attempts to choose an image that best matches your target platform. In addition, if you run **eb init** in interactive mode and it fails to choose an image for you, it prompts you to choose one. At the end of a successful initialization, **eb init** writes the chosen image into the `buildspec.yml` file.

`Timeout`

The duration, in minutes, that the CodeBuild build runs before timing out. This value is optional. For details about valid and default values, see Create a Build Project in CodeBuild.

Note

This timeout controls the maximum duration for a CodeBuild run, and the EB CLI also respects it as part of its first step to create an application version. It's distinct from the value you can specify with the `--timeout` option of the **eb create** or **eb deploy** commands. The latter value controls the maximum duration that for EB CLI to wait for environment creation or update.

Building and deploying your application code

Whenever your application code needs to be deployed, the EB CLI uses CodeBuild to run a build, then deploys the resulting build artifacts to your environment. This happens when you create an Elastic Beanstalk environment for your application using the **eb create** command, and each time you later deploy code changes to the environment using the **eb deploy** command.

If the CodeBuild step fails, environment creation or deployment doesn't start.

Tagging applications

This topic explains the benefits of tagging your Elastic Beanstalk applications. It also provides instructions to create and manage application tags. Tags are key-value pairs associated with AWS resources. For information about Elastic Beanstalk resource tagging, use cases, tag key and value constraints, and supported resource types, see Tagging Elastic Beanstalk application resources.

You can specify tags when you create an application. In an existing application, you can add or remove tags, and update the values of existing tags. You can add up to 50 tags to each application.

Adding tags during application creation

When you use the Elastic Beanstalk console to create an application, you can specify tag keys and values in the **Create New Application** dialog box.

If you use the EB CLI to create an application, use the `--tags` option with **eb init** to add tags.

```
~/workspace/my-app$ eb init --tags mytag1=value1,mytag2=value2
```

With the AWS CLI or other API-based clients, add tags by using the `--tags` parameter on the **create-application** command.

```
$ aws elasticbeanstalk create-application \
      --tags Key=mytag1,Value=value1 Key=mytag2,Value=value2 \
      --application-name my-app --version-label v1
```

Managing tags of an existing application

You can add, update, and delete tags in an existing Elastic Beanstalk application.

To manage an application's tags in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.
3. Choose **Actions**, and then choose **Manage tags**.
4. Use the on-screen form to add, update, or delete tags.

5. To save the changes choose **Apply** at the bottom of the page.

If you use the EB CLI to update your application, use **eb tags** to add, update, delete, or list tags.

For example, the following command lists the tags in an application.

```
~/workspace/my-app$ eb tags --list --resource "arn:aws:elasticbeanstalk:us-east-2:my-
account-id:application/my-app"
```

The following command updates the tag `mytag1` and deletes the tag `mytag2`.

```
~/workspace/my-app$ eb tags --update mytag1=newvalue --delete mytag2 \
      --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:application/my-app"
```

For a complete list of options and more examples, see `eb tags`.

With the AWS CLI or other API-based clients, use the **list-tags-for-resource** command to list the tags of an application.

```
$ aws elasticbeanstalk list-tags-for-resource --resource-arn
 "arn:aws:elasticbeanstalk:us-east-2:my-account-id:application/my-app"
```

Use the **update-tags-for-resource** command to add, update, or delete tags in an application.

```
$ aws elasticbeanstalk update-tags-for-resource \
      --tags-to-add Key=mytag1,Value=newvalue --tags-to-remove mytag2 \
      --resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-id:application/my-
app"
```

Specify both tags to add and tags to update in the `--tags-to-add` parameter of **update-tags-for-resource**. A nonexisting tag is added, and an existing tag's value is updated.

Note

To use some of the EB CLI and AWS CLI commands with an Elastic Beanstalk application, you need the application's ARN. You can retrieve the ARN by using the following command.

```
$ aws elasticbeanstalk describe-applications --application-names my-app
```

Tagging Elastic Beanstalk application resources

This topic explains the benefits of using tags with your Elastic Beanstalk application resources along with the constraints of doing so. It also explains how to create and manage tags for application resources.

You can apply tags to resources of your AWS Elastic Beanstalk applications. Tags are key-value pairs associated with AWS resources. Tags can help you categorize resources. They're particularly useful if you manage many resources as part of multiple AWS applications.

Here are some ways to use tagging with Elastic Beanstalk resources:

- *Deployment stages* – Identify resources associated with different stages of your application, such as development, beta, and production.
- *Cost allocation* – Use cost allocation reports to track your usage of AWS resources associated with various expense accounts. The reports include both tagged and untagged resources, and they aggregate costs according to tags. For information about how cost allocation reports use tags, see Use Cost Allocation Tags for Custom Billing Reports in the *AWS Billing and Cost Management User Guide*.
- *Access control* – Use tags to manage permissions to requests and resources. For example, a user who can only create and manage beta environments should only have access to beta stage resources. For details, see Using tags to control access to Elastic Beanstalk resources.

You can add up to 50 tags to each resource. Environments are slightly different: Elastic Beanstalk adds three default system tags to environments, and you can't edit or delete these tags. In addition to the default tags, you can add up to 47 additional tags to each environment.

The following constraints apply to tag keys and values:

- Keys and values can contain letters, numbers, white space, and the following symbols: `_ . : / = + - @`
- Keys can contain up to 127 characters. Values can contain up to 255 characters.

Note

These length limits are for Unicode characters in UTF-8. For other multibyte encodings, the limits might be lower.

- Keys are case sensitive.
- Keys cannot begin with `aws:` or `elasticbeanstalk:`.

Resources you can tag

The following are the types of Elastic Beanstalk resources that you can tag, and links to specific topics about managing tags for each of them:

- Applications
- Environments
- Application versions
- Saved configurations

Tag propagation to launch templates

Elastic Beanstalk provides an option to enable the propagation of environment tags to launch templates. This option provides continued support for tag-based access control (TBAC) with launch templates.

Note

Launch configurations are being phased out and replaced by launch templates. For more information, see Launch configurations in the *Amazon EC2 Auto Scaling User Guide*.

To prevent down-time of running EC2 instances AWS CloudFormation doesn't propagate tags to existing launch templates. If there's a use case that requires tags for your environment's resources, you can enable Elastic Beanstalk to create launch templates with tags for these resources. To do so, set the `LaunchTemplateTagPropagationEnabled` option in the aws:autoscaling:launchconfiguration namespace to `true`. The default value is `false`.

The following configuration file example enables the propagation of tags to launch templates.

```
option_settings:
  aws:autoscaling:launchconfiguration:
    LaunchTemplateTagPropagationEnabled: true
```

Elastic Beanstalk can only propagate tags to launch templates for the following resources:

- EBS volumes
- EC2 instances
- EC2 network interfaces
- AWS CloudFormation launch templates that define a resource

This constraint exists because CloudFormation only allows tags on template creation for specific resources. For more information see TagSpecification in the *AWS CloudFormation User Guide*.

Important

- Changing this option value from `false` to `true` for an existing environment may be a breaking change for previously existing tags.
- When this feature is enabled, the propagation of tags will require EC2 replacement, which can result in downtime. You can enable *rolling updates* to apply configuration changes in batches and prevent downtime during the update process. For more information, see Configuration changes.

For more information about launch templates, see the following:

- Launch templates in the *Amazon EC2 Auto Scaling User Guide*
- Working with templates in the *AWS CloudFormation User Guide*
- Elastic Beanstalk template snippets in the *AWS CloudFormation User Guide*

Creating environments in Elastic Beanstalk

This chapter describes how to create and manage your Elastic Beanstalk environments. This introductory page provides an overview of updates, maintenance, and configurations that you'll apply over time as your application and environment evolve.

Environment functions

You can create and manage separate environments for development, testing, and production use, and you can deploy any version of your application to any environment. Environments can be long-running or temporary. When you terminate an environment, you can save its configuration to recreate it later.

Application deployments

As you develop your application, you will deploy it often, possibly to several different environments for different purposes. Elastic Beanstalk lets you configure how deployments are performed. You can deploy to all of the instances in your environment simultaneously, or split a deployment into batches with rolling deployments.

Configuration changes

Configuration changes are processed separately from deployments, and have their own scope. For example, if you change the type of the EC2 instances running your application, all of the instances must be replaced. On the other hand, if you modify the configuration of the environment's load balancer, that change can be made in-place without interrupting service or lowering capacity. You can also apply configuration changes that modify the instances in your environment in batches with rolling configuration updates.

Note

Modify the resources in your environment only by using Elastic Beanstalk. If you modify resources using another service's console, CLI commands, or SDKs, Elastic Beanstalk won't be able to accurately monitor the state of those resources, and you won't be able to save the configuration or reliably recreate the environment. Out-of band-changes can also cause issues when updating or terminating an environment.

Platform updates

When you launch an environment, you choose a platform version. We update platforms periodically with new platform versions to provide performance improvements and new features. You can update your environment to the latest platform version at any time. See the *AWS Elastic Beanstalk Platforms* guide for a list of supported platforms and a platform version history that includes the date ranges they were current.

Architecture options

As your application grows in complexity, you can split it into multiple components, each running in a separate environment. For long-running workloads, you can launch worker environments that process jobs from an Amazon Simple Queue Service (Amazon SQS) queue.

Topics

Using the Elastic Beanstalk environment management console

This section describes how you can manage your Elastic Beanstalk environment using the environment management console. The console provides the capability to manage your environment's configuration and perform common actions. These actions include restarting the web servers running in your environment, cloning your environment, or rebuilding your environment from scratch.

Topics

Accessing the environment management console

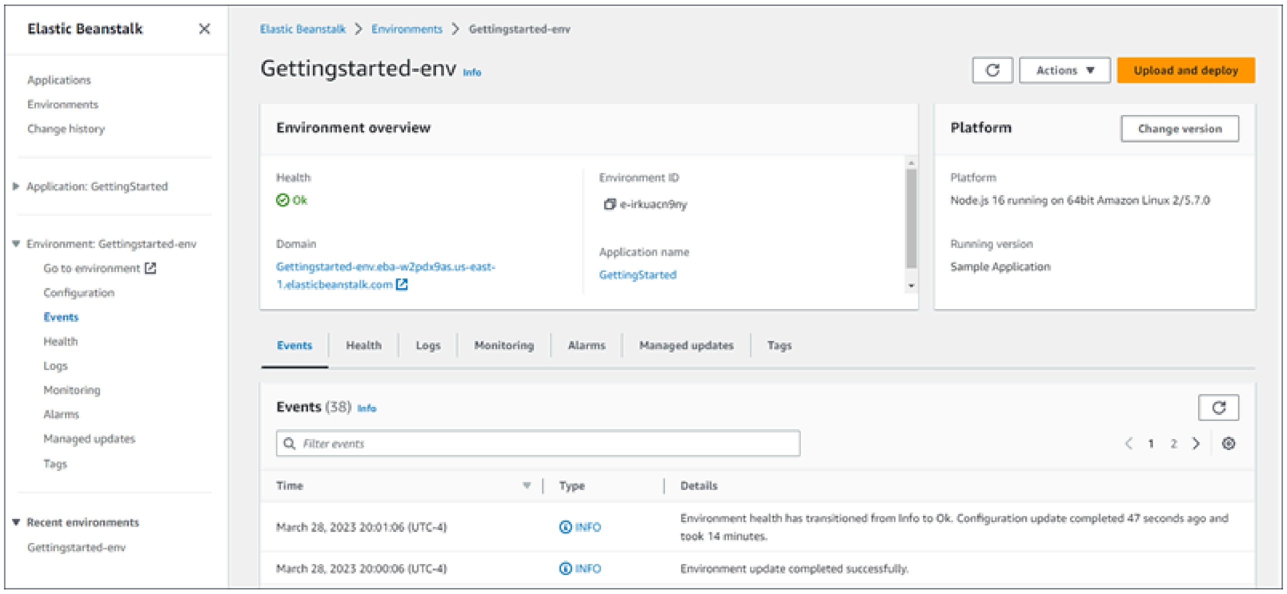
The following procedure provides steps to launch the environment management console.

If you're already logged in to the Elastic Beanstalk console, you can also launch the environment management page from the Application management console. Select an environment from the list to display the management console details for the selected environment.

To access the environment management console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

The following image illustrates the environment management console.



Health: Ok

Environment ID: e-irkuacn9ny

Domain: Gettingstarted-env.eba-w2pdx9as.us-east-1.elasticbeanstalk.com

Application name: GettingStarted

Platform: Node.js 16 running on 64bit Amazon Linux 2/5.7.0

Running version: Sample Application

Time	Type	Details
March 28, 2023 20:01:06 (UTC-4)	INFO	Environment health has transitioned from Info to Ok. Configuration update completed 47 seconds ago and took 14 minutes.
March 28, 2023 20:00:06 (UTC-4)	INFO	Environment update completed successfully.

The top pane is the **Environment overview** page. It shows top-level information about your environment.

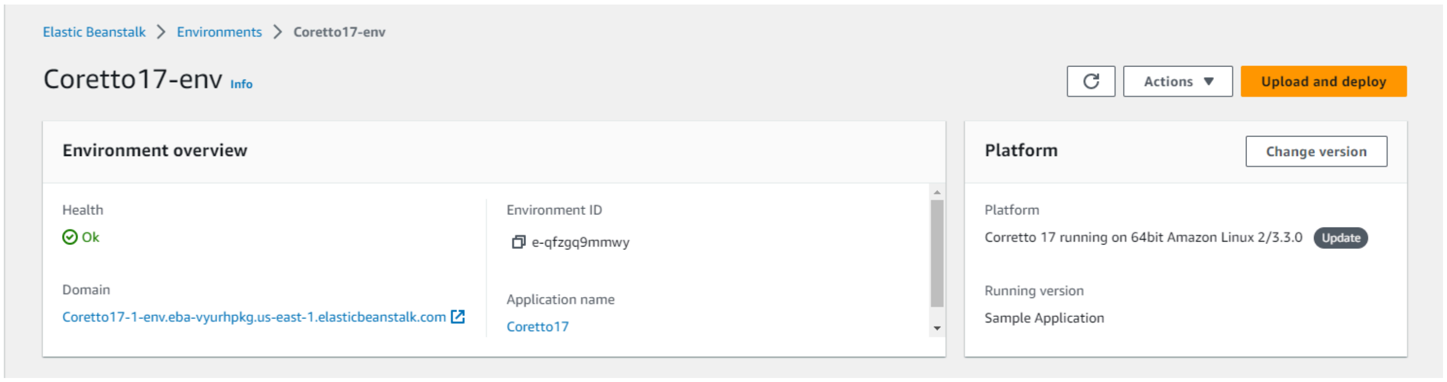
The bottom half of the page displays tabs that provide more detailed information. The **Events** tab displays by default. The pages that are linked to the tabs, are also listed on the left navigation pane under the environment.

The console's navigation pane shows the name of the application that's deployed to the environment, with related application management pages. The environment name is also displayed on the navigation page, followed by the environment management pages. The links listed under the environment name also include **Go to environment** and **Configuration**, in addition to the tabbed pages previously mentioned.

Environment overview pane

This topic describes the information that the **Environment overview** pane provides. It shows top-level information about your environment and is located on the top half of the environment management console.

The following image displays the **Environment overview** pane.



Environment overview

Health: Ok
Domain: Coretto17-1-env.eba-vyurhpkg.us-east-1.elasticbeanstalk.com
Environment ID: e-qfzgq9mmwy
Application name: Coretto17

Platform

Platform: Corretto 17 running on 64bit Amazon Linux 2/3.3.0
Running version: Sample Application

Health

The overall health of the environment. If the health of your environment degrades, the **View causes** link displays next to the environment health. Select this link to view the **Health** tab with more details.

Domain

The environment's **Domain**, or URL, is located in the upper portion of the **Environment overview** page, below the environment's **Health**. This is the URL of the web application that the environment is running. You can launch the application be selecting the URL.

Environment id

The environment ID. This is an internal ID that's generated when the environment is created.

Application name

The name of the application that is deployed and running on your environment.

Running version

The name of the application version that is deployed and running on your environment. Choose **Upload and deploy** to upload a source bundle and deploy it to your environment. This option creates a new application version.

Platform

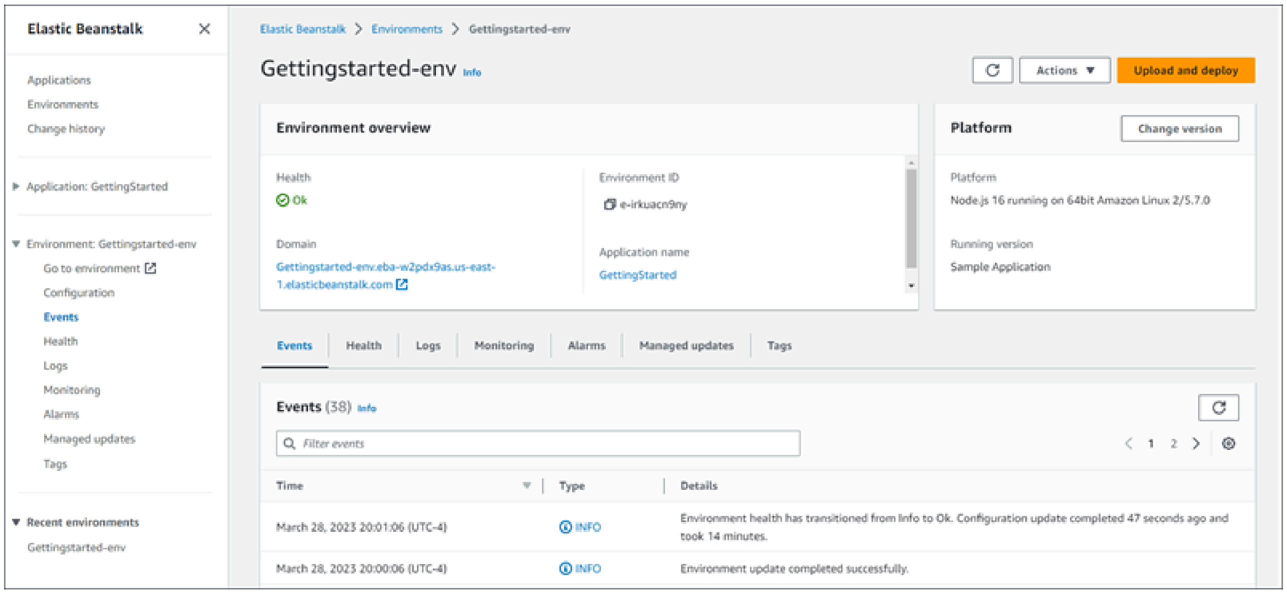
The name of the platform version running on your environment. Typically, this comprises the architecture, operating system (OS), language, and application server (collectively known as the *platform branch*), with a specific platform version number.

If your platform version is not the most recently available, then a status label displays next to it in the **Platform** section. The **Update** label indicates that although the platform version is supported a newer version is available. The platform version may also be labeled as **Deprecated** or **Retired**. Select **Change version** to update your platform branch to a newer version. For more information about the *states* of a platform version, see the *Platform Branch* section in the Elastic Beanstalk platforms glossary. The previous image on this page illustrates the **Update** status label for the given platform.

Environment detail

This topic describes the additional information that the environment management console provides from the left navigation pane and the tabbed pages.

The following image illustrates the environment management console.



Environment overview

Health: Ok

Environment ID: e-irkuacn9ny

Domain: Gettingstarted-env.eba-w2pdx9as.us-east-1.elasticbeanstalk.com

Application name: GettingStarted

Platform

Platform: Node.js 16 running on 64bit Amazon Linux 2/5.7.0

Running version: Sample Application

Events (38) Info

Time	Type	Details
March 28, 2023 20:01:06 (UTC-4)	INFO	Environment health has transitioned from Info to Ok. Configuration update completed 47 seconds ago and took 14 minutes.
March 28, 2023 20:00:06 (UTC-4)	INFO	Environment update completed successfully.

The bottom half of the environment management console lists tabs that provide more detailed and varied information about the environment. You can either select the tab page or the page label from the left navigation pane.

From the left navigation pane of the console under the environment name, there are two choices that are not in the tabbed pages. These are **Go to environment** and **Configuration**.

> **Note**
>
> Select **Go to environment** to launch your application.

Configuration

Use the **Configuration** page on the left navigation pane to view and update current configuration settings for your environment and its resources. This includes networking configuration, database configuration, load balancing, notifications, health monitoring settings, managed platform update configuration, deployment configuration, instance log streaming, CloudWatch integration, AWS X-Ray, proxy server settings, environment properties, and platform specific options. Use the settings on this page to customize the behavior of your environment during deployments, enable additional features, and modify the instance type and other settings that you chose during environment creation.

For more information, see Configuring Elastic Beanstalk environments.

Events

The **Events** page shows the event stream for your environment. Elastic Beanstalk outputs event messages whenever you interact with the environment, and when any of your environment's resources are created or modified as a result.

For more information, see Viewing an Elastic Beanstalk environment's event stream.

Health

If enhanced health monitoring is enabled, this page lists the EC2 instances in your environment and the live health information for each instance.

The **Overall health** page shows health data as an average for all of your environment’s instances combined.

The **Enhanced instance health** pane shows live health information for each individual EC2 instance in your environment. Enhanced health monitoring enables Elastic Beanstalk to closely monitor the resources in your environment so that it can assess the health of your application more accurately.

When enhanced health monitoring is enabled, this page shows information about the requests served by the instances in your environment and metrics from the operating system, including latency, load, and CPU utilization.

For more information, see Enhanced health reporting and monitoring in Elastic Beanstalk.

Logs

The **Logs** page lets you retrieve logs from the EC2 instances in your environment. When you request logs, Elastic Beanstalk sends a command to the instances, which then upload logs to your Elastic Beanstalk storage bucket in Amazon S3. When you request logs on this page, Elastic Beanstalk automatically deletes them from Amazon S3 after 15 minutes.

You can also configure your environment's instances to upload logs to Amazon S3 for permanent storage after they have been rotated locally.

For more information, see Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment.

Monitoring

The **Monitoring** page shows an overview of health information for your environment. This includes the default set of metrics provided by Elastic Load Balancing and Amazon EC2, and graphs that show how the environment's health has changed over time.

For more information, see Monitoring environment health in the AWS management console.

Alarms

The **Existing alarms** page shows information about any alarms that you have configured for your environment. You can use the options on this page to create or delete alarms.

For more information, see Manage alarms.

Managed updates

The **Managed updates** page shows information about upcoming and completed managed platform updates and instance replacement.

The managed update feature lets you configure your environment to update to the latest platform version automatically during a weekly maintenance window that you choose. In between platform releases, you can choose to have your environment replace all of its Amazon EC2 instances during the maintenance window. This can alleviate issues that occur when your application runs for extended periods of time.

For more information, see Managed platform updates.

Tags

The **Tags** page shows the tags that Elastic Beanstalk applied to the environment when you created it, and any tags that you added. You can add, edit, and delete custom tags. You can't edit or delete the tags that Elastic Beanstalk applied.

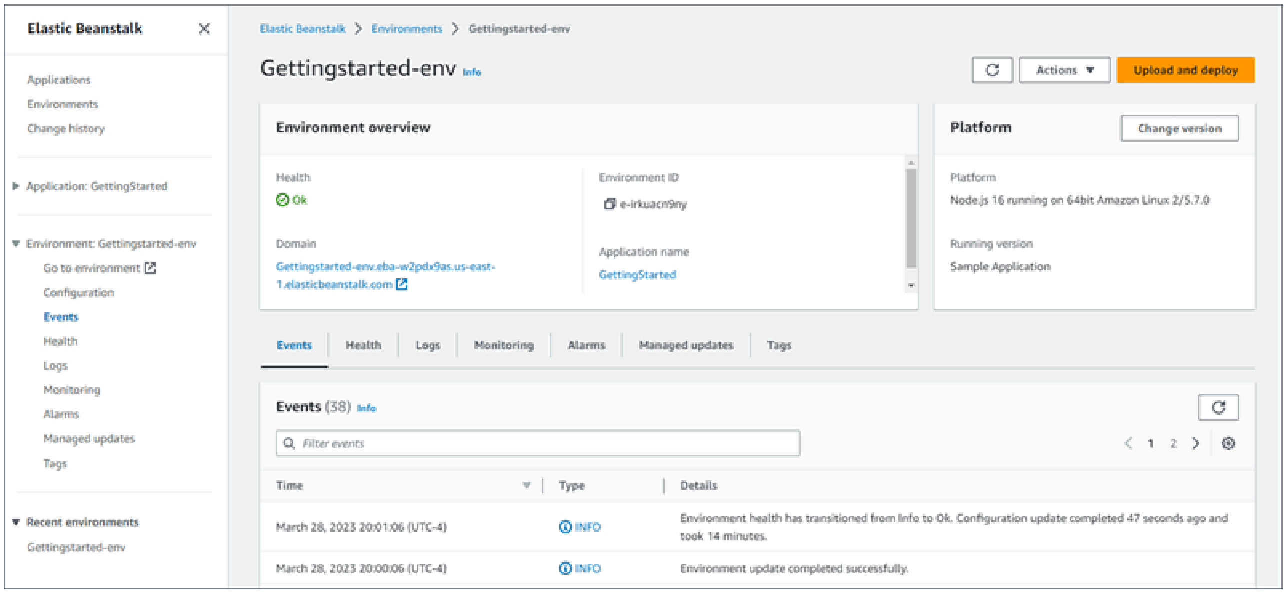
Environment tags are applied to every resource that Elastic Beanstalk creates to support your application.

For more information, see Tagging resources in your Elastic Beanstalk environments.

Environment actions

This topic describes the common operations that you can select to perform on your environment from the **Actions** drop-down menu on the environment management console.

The following image illustrates the environment management console. The **Actions** drop-down menu is on the right side of the header that displays the environment name, next to the **Refresh** button.



Health: Ok
Environment ID: e-irkuacn9ny
Domain: Gettingstarted-env.eba-w2pdx9as.us-east-1.elasticbeanstalk.com
Application name: GettingStarted
Platform: Node.js 16 running on 64bit Amazon Linux 2/5.7.0
Running version: Sample Application

Time	Type	Details
March 28, 2023 20:01:06 (UTC-4)	INFO	Environment health has transitioned from Info to Ok. Configuration update completed 47 seconds ago and took 14 minutes.
March 28, 2023 20:00:06 (UTC-4)	INFO	Environment update completed successfully.

Note

Some actions are only available under certain conditions, remaining disabled until the right conditions are met.

Load configuration

Load a previously saved configuration. Configurations are saved to your application and can be loaded by any associated environment. If you've made changes to your environment's configuration, you can load a saved configuration to undo those changes. You can also load a configuration that you saved from a different environment running the same application to propagate configuration changes between them.

Save configuration

Save the current configuration of your environment to your application. Before you make changes to your environment's configuration, save the current configuration so that you can roll back later, if needed. You can also apply a saved configuration when you launch a new environment.

Swap environment Domains (URLs)

Swap the CNAME of the current environment with a new environment. After a CNAME swap, all traffic to the application using the environment URL goes to the new environment. When you are ready to deploy a new version of your application, you can launch a separate environment under the new version. When the new environment is ready to start taking requests, perform a CNAME swap to start routing traffic to the new environment. Doing this doesn't interrupt your services. For more information, see Blue/Green deployments with Elastic Beanstalk.

Clone environment

Launch a new environment with the same configuration as your currently running environment.

Clone with latest platform

Clone your current environment with the latest version of the in-use Elastic Beanstalk platform. This option is available only when a newer version of the current environment's platform is available for use.

Abort current operation

Stop an in-progress environment update. Stopping an operation can cause some of the instances in your environment to be in a different state than others, depending on how far the operation progressed. This option is available only when your environment is being updated.

Restart app servers

Restart the web server that is running on your environment's instances. This option doesn't terminate or restart any AWS resources. If your environment is acting strangely in response to some bad requests, restarting the application server can restore functionality temporarily while you troubleshoot the root cause.

Rebuild environment

Terminate all resources in the running environment and build a new environment with the same settings. This operation takes several minutes, similar to the amount of time needed for deploying a new environment from scratch. Any Amazon RDS instances that are running in your environment's data tier are deleted during a rebuild. If you need the data, create a snapshot. You can create a snapshot manually in the RDS console or configure your data tier's Deletion Policy to create a snapshot automatically before deleting the instance. This is the default setting when you create a data tier.

Terminate environment

Terminate all resources in the running environment and remove the environment from the application. If you have an RDS instance that is running in a data tier and you need to retain its data, make sure the *database deletion policy* is set to either `Snapshot` or `Retain`. For more information, see Database lifecycle in the *Configuring environments* chapter of this guide.

Creating an Elastic Beanstalk environment

The following procedure launches a new environment running the default application. These steps are simplified to get your environment up and running quickly, using default option values.

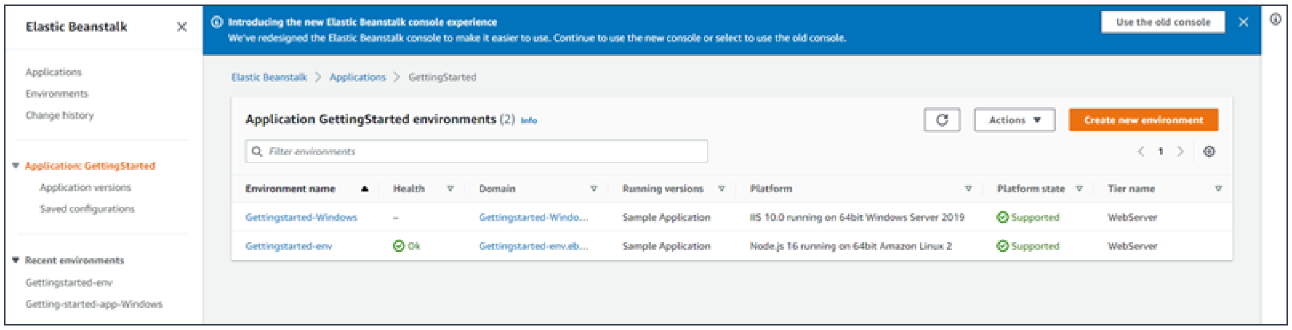
Note about permissions

Creating an environment requires the permissions in the Elastic Beanstalk full access managed policy. See Elastic Beanstalk user policy for details.

To launch an environment with a sample application (console)

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**. Select an existing application in the list. You can also choose to create one, following the instructions in Managing applications .
3. On the application overview page, choose **Create new environment**.

   The following image displays the application overview page.



Introducing the new Elastic Beanstalk console experience
We've redesigned the Elastic Beanstalk console to make it easier to use. Continue to use the new console or select to use the old console.

Application GettingStarted environments (2) Info

Environment name	Health	Domain	Running versions	Platform	Platform state	Tier name
Gettingstarted-Windows	-	Gettingstarted-Windo...	Sample Application	IIS 10.0 running on 64bit Windows Server 2019	Supported	WebServer
Gettingstarted-env	Ok	Gettingstarted-env.eb...	Sample Application	Node.js 16 running on 64bit Amazon Linux 2	Supported	WebServer

This launches the **Create environment** wizard. The wizard provides a set of steps for you to create a new environment.

4. For **Environment tier**, choose the **Web server environment** or **Worker environment** environment tier. You can't change an environment's tier after creation.

> **Note**
>
> The .NET on Windows Server platform doesn't support the worker environment tier.

The **Application information** fields default, based on the application that you previously chose.

In the **Environment information** grouping the **Environment name** defaults, based on the application name. If you prefer a different environment name you can enter another value in the field. You can optionally enter a **Domain** name; otherwise Elastic Beanstalk autogenerates a value. You can also optionally enter an **Environment description.**

5. For **Platform**, select the platform and platform branch that match the language your application uses.

> **Note**
>
> Elastic Beanstalk supports multiple versions for most of the platforms that are listed. By default, the console selects the recommended version for the platform and platform branch you choose. If your application requires a different version, you can select it here. For information about supported platform versions, see the section called "Supported platforms".

6. For **Application code**, you have some choices for launching a sample application.

- To launch the default sample application without supplying the source code, choose **Sample application**. This action chooses the single page application that Elastic Beanstalk provides for the platform you previously selected.
- If you downloaded a sample application from this guide or another source, do the following steps.

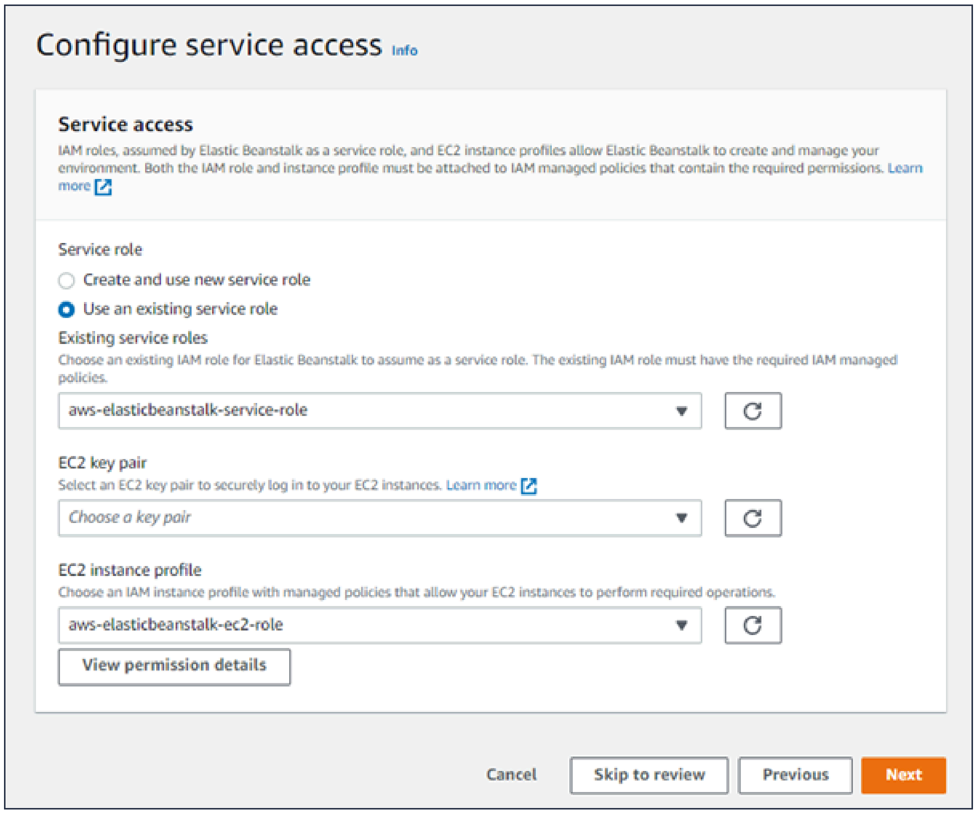
   a. Select **Upload your code**.

   b. Next choose **Local file**, then under **Upload application**, select **Choose file**.

   c. Your computer's operating system will present you with an interface to select the local file that you downloaded. Select the source bundle file and continue.

7. For **Presets**, choose **Single instance**.
8. Choose **Next**.
9. The **Configure service access** page displays.

   The following image illustrates the **Configure service access** page.



Configure service access Info

Service access

IAM roles, assumed by Elastic Beanstalk as a service role, and EC2 instance profiles allow Elastic Beanstalk to create and manage your environment. Both the IAM role and instance profile must be attached to IAM managed policies that contain the required permissions. Learn more

Service role

Create and use new service role [✓ Use an existing service role]

Existing service roles

Choose an existing IAM role for Elastic Beanstalk to assume as a service role. The existing IAM role must have the required IAM managed policies.

aws-elasticbeanstalk-service-role

EC2 key pair

Select an EC2 key pair to securely log in to your EC2 instances. Learn more

Choose a key pair

EC2 instance profile

Choose an IAM instance profile with managed policies that allow your EC2 instances to perform required operations.

aws-elasticbeanstalk-ec2-role

10. Choose a value from the **Existing Service Roles** dropdown.
11. (Optional) If you previously created an EC2 key pair, you can select it from the **EC2 key pair** field dropdown. You would use it to securely log in to the Amazon EC2 instance that Elastic

Beanstalk provisions for your application. If you skip this step, you can always create and assign an EC2 key pair after the environment is created. For more information, see EC2 key pair.

12. Next, we'll focus on the **EC2 instance profile** dropdown list. The values displayed in this dropdown list may vary, depending on whether you account has previously created a new environment.

    Choose one of the following items, based on the values displayed in your list.

    - If `aws-elasticbeanstalk-ec2-role` displays in the dropdown list, select it from the dropdown list.
    - If another value displays in the list, and it’s the default EC2 instance profile intended for your environments, select it from the dropdown list.
    - If the **EC2 instance profile** dropdown list doesn't list any values, you'll need to create an instance profile.

      Create an instance profile

      To create an instance profile, we'll take a detour to another procedure on this same page. Go to the end of this procedure and expand the procedure that follows, *Create IAM Role for EC2 instance profile*.
      Complete the steps in *Create IAM Role for EC2 instance profile* to create an IAM Role that you can subsequently select for the EC2 instance profile. Then return back to this step.

      Now that you've created an IAM Role, and refreshed the list, it displays as a choice in the dropdown list. Select the IAM Role you just created from the **EC2 instance profile** dropdown list.

13. Choose **Skip to Review** on the **Configure service access** page.

    This will select the default values for this step and skip the optional steps.

14. The **Review** page displays a summary of all your choices.

    To further customize your environment, choose **Edit** next to the step that includes any items you want to configure. You can set the following options only during environment creation:

- Environment name
- Domain name
- Platform version
- Processor
- VPC
- Tier

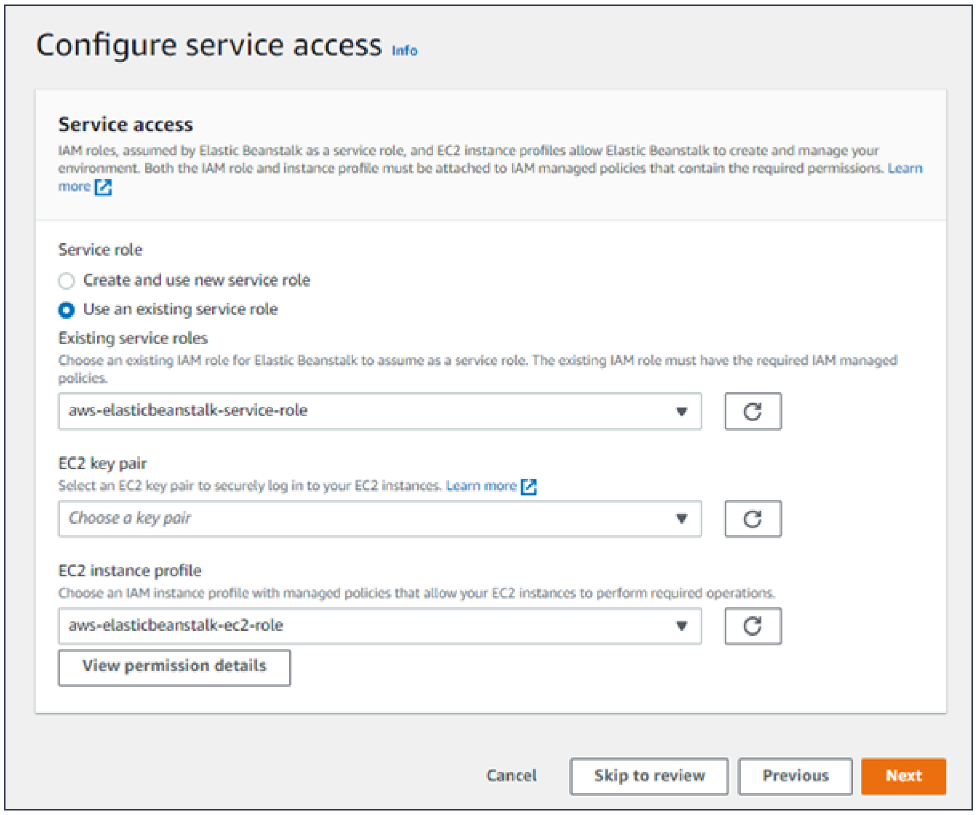
You can change the following settings after environment creation, but they require new instances or other resources to be provisioned and can take a long time to apply:

- Instance type, root volume, key pair, and AWS Identity and Access Management (IAM) role
- Internal Amazon RDS database
- Load balancer

For details on all available settings, see The create new environment wizard.

15. Choose **Submit** at the bottom of the page to initialize the creation of your new environment.

Create IAM Role for EC2 instance profile



Configure service access Info

Service access

IAM roles, assumed by Elastic Beanstalk as a service role, and EC2 instance profiles allow Elastic Beanstalk to create and manage your environment. Both the IAM role and instance profile must be attached to IAM managed policies that contain the required permissions. Learn more

Service role

○ Create and use new service role

● Use an existing service role

Existing service roles

Choose an existing IAM role for Elastic Beanstalk to assume as a service role. The existing IAM role must have the required IAM managed policies.

aws-elasticbeanstalk-service-role

EC2 key pair

Select an EC2 key pair to securely log in to your EC2 instances. Learn more

Choose a key pair

EC2 instance profile

Choose an IAM instance profile with managed policies that allow your EC2 instances to perform required operations.

aws-elasticbeanstalk-ec2-role

To create the EC2 instance profile

1. Choose **Create role**.
2. For **Trusted entity type**, choose **AWS service**.
3. For **Use case**, choose **Elastic Beanstalk – Compute**.
4. Choose **Next**.
5. Verify that **Permissions policies** include the following, then choose **Next**:
   - `AWSElasticBeanstalkWebTier`
   - `AWSElasticBeanstalkWorkerTier`
   - `AWSElasticBeanstalkMulticontainerDocker`
6. Choose **Create role**.
7. Return to the **Configure service access** tab, refresh the list, then select the newly created EC2 instance profile.

While Elastic Beanstalk creates your environment, you are redirected to the Elastic Beanstalk console. When the environment health turns green, choose the URL next to the environment name to view the running application. This URL is generally accessible from the internet unless you configure your environment to use a custom VPC with an internal load balancer.

Topics

The create new environment wizard

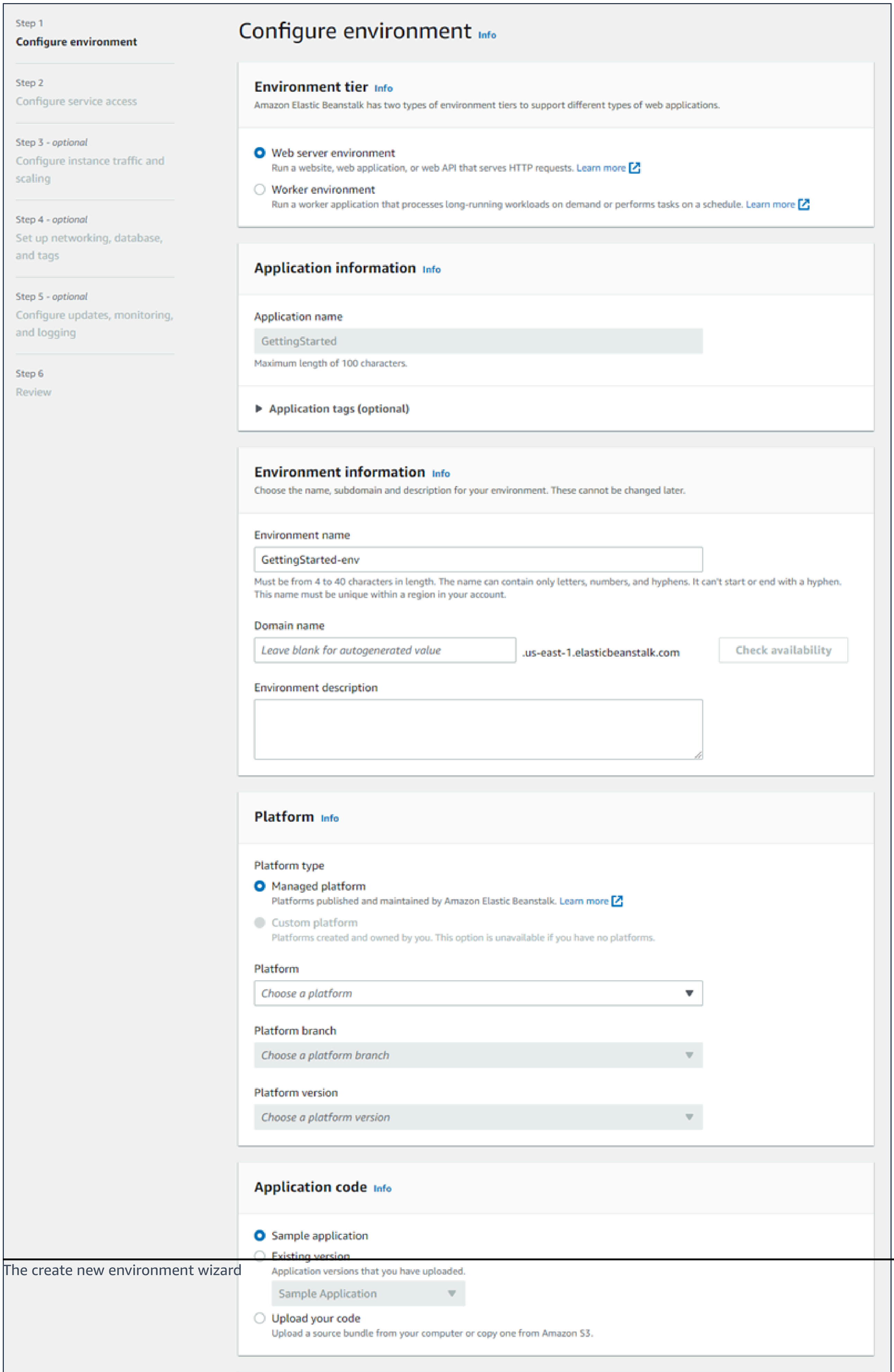
This topic describes the **Create environment** wizard and all the ways you can use it to configure the environment you want to create.

Note

In Creating an Elastic Beanstalk environment we show how to launch the **Create environment** wizard and quickly create an environment with default values and recommended settings. This current topic will walk you through all of the options.

Wizard page

The **Create environment** wizard provides a set of steps for you to create a new environment.



Step 1
Configure environment

Step 2
Configure service access

Step 3 - *optional*
Configure instance traffic and scaling

Step 4 - *optional*
Set up networking, database, and tags

Step 5 - *optional*
Configure updates, monitoring, and logging

Step 6
Review

Configure environment Info

Environment tier Info
Amazon Elastic Beanstalk has two types of environment tiers to support different types of web applications.

[✓ Web server environment]
Run a website, web application, or web API that serves HTTP requests. Learn more

Worker environment
Run a worker application that processes long-running workloads on demand or performs tasks on a schedule. Learn more

Application information Info

Application name
GettingStarted
Maximum length of 100 characters.

▶ Application tags (optional)

Environment information Info
Choose the name, subdomain and description for your environment. These cannot be changed later.

Environment name
GettingStarted-env
Must be from 4 to 40 characters in length. The name can contain only letters, numbers, and hyphens. It can't start or end with a hyphen. This name must be unique within a region in your account.

Domain name
Leave blank for autogenerated value .us-east-1.elasticbeanstalk.com Check availability

Environment description

Platform Info

Platform type
[✓ Managed platform]
Platforms published and maintained by Amazon Elastic Beanstalk. Learn more

Custom platform
Platforms created and owned by you. This option is unavailable if you have no platforms.

Platform
Choose a platform

Platform branch
Choose a platform branch

Platform version
Choose a platform version

Application code Info

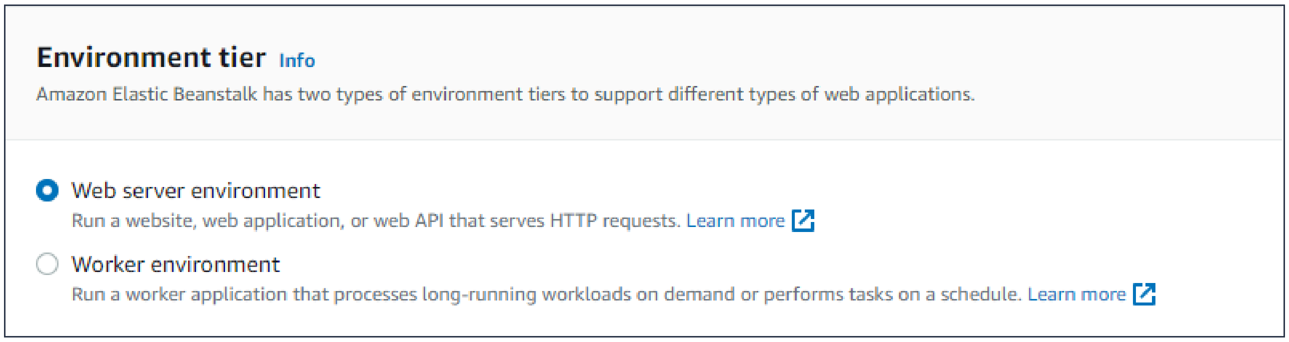
[✓ Sample application]

Existing version
Application versions that you have uploaded.
Sample Application

Upload your code
Upload a source bundle from your computer or copy one from Amazon S3.

Environment tier

For **environment tier**, choose the **Web server environment** or **Worker environment** environment tier. You can't change an environment's tier after creation.



Environment tier Info

Amazon Elastic Beanstalk has two types of environment tiers to support different types of web applications.

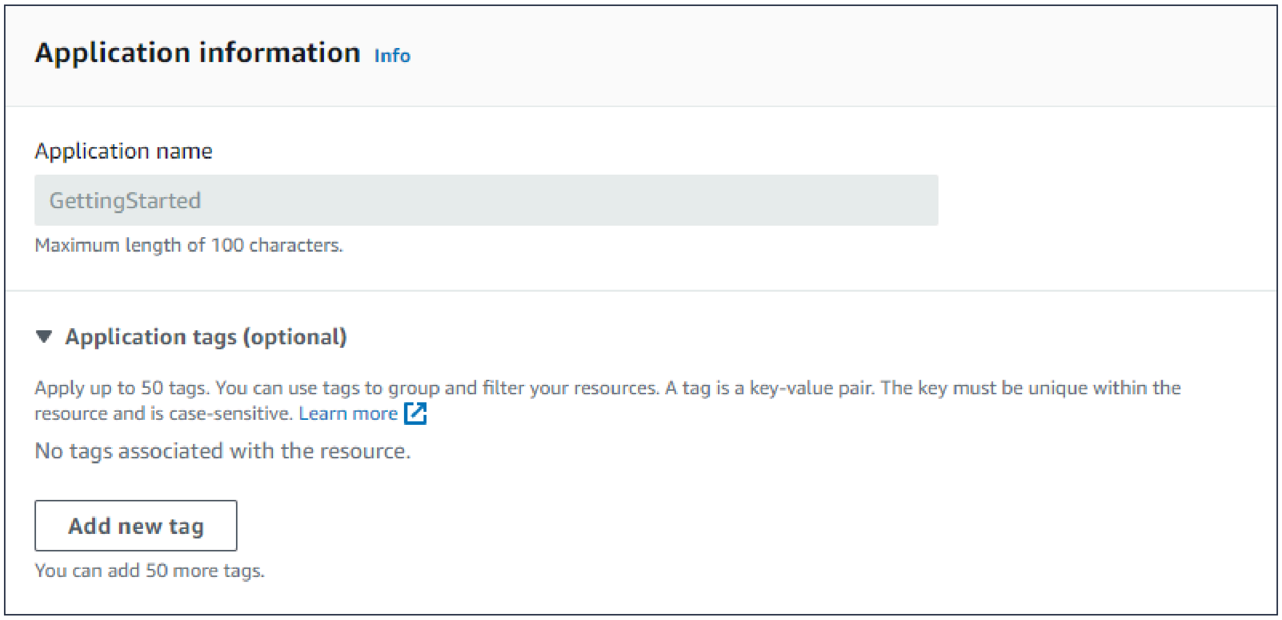
[✓ Web server environment]
Run a website, web application, or web API that serves HTTP requests. Learn more

Worker environment
Run a worker application that processes long-running workloads on demand or performs tasks on a schedule. Learn more

> **Note**
>
> The .NET on Windows Server platform doesn't support the worker environment tier.

Application information

If you launched the wizard by selecting **Create new environment** from the **Application overview** page, then the **Application name** is prefilled. Otherwise, enter an application name. Optionally, add application tags.



Application information Info

Application name

GettingStarted

Maximum length of 100 characters.

▼ Application tags (optional)

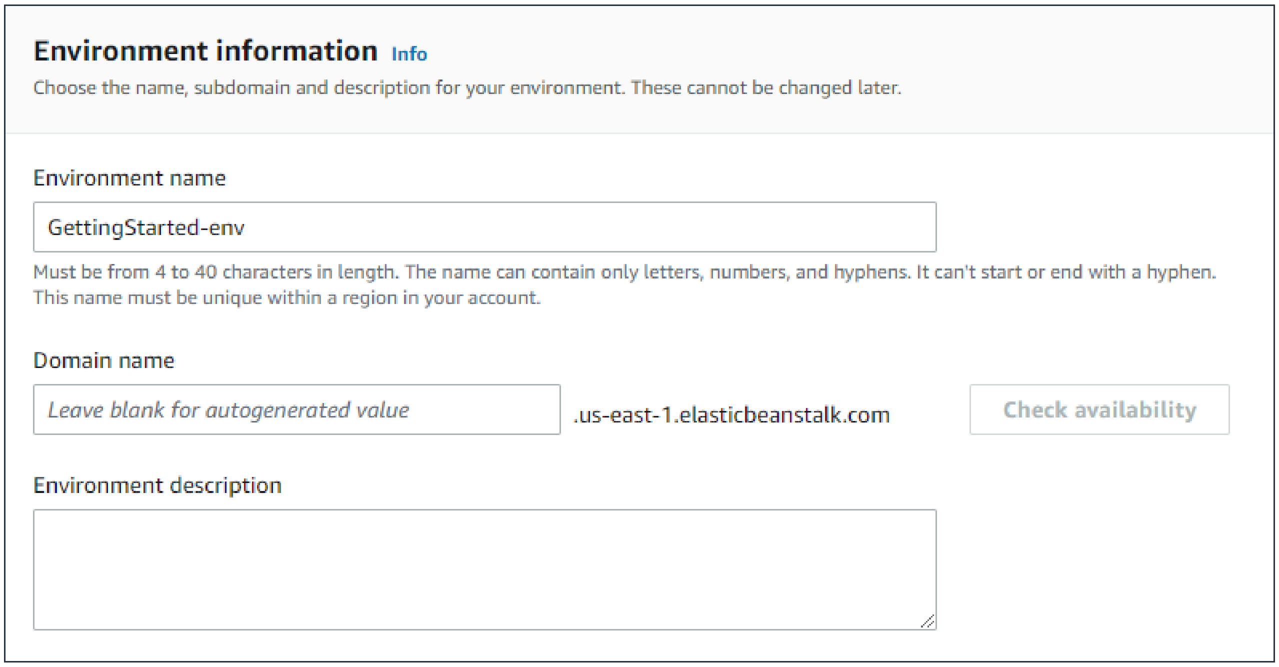
Apply up to 50 tags. You can use tags to group and filter your resources. A tag is a key-value pair. The key must be unique within the resource and is case-sensitive. Learn more

No tags associated with the resource.

You can add 50 more tags.

Environment information

Set the environment's name and domain, and create a description for your environment. Be aware that these environment settings cannot change after the environment is created.



Environment information Info

Choose the name, subdomain and description for your environment. These cannot be changed later.

Environment name

GettingStarted-env

Must be from 4 to 40 characters in length. The name can contain only letters, numbers, and hyphens. It can't start or end with a hyphen. This name must be unique within a region in your account.

Domain name

Leave blank for autogenerated value .us-east-1.elasticbeanstalk.com

Environment description

- **Name –** Enter a name for the environment. The form provides a generated name.
- **Domain –** (web server environments) Enter a unique domain name for your environment. The default name is the environment's name. You can enter a different domain name. Elastic Beanstalk uses this name to create a unique CNAME for the environment. To check whether the domain name you want is available, choose **Check Availability**.
- **Description –** Enter a description for this environment.

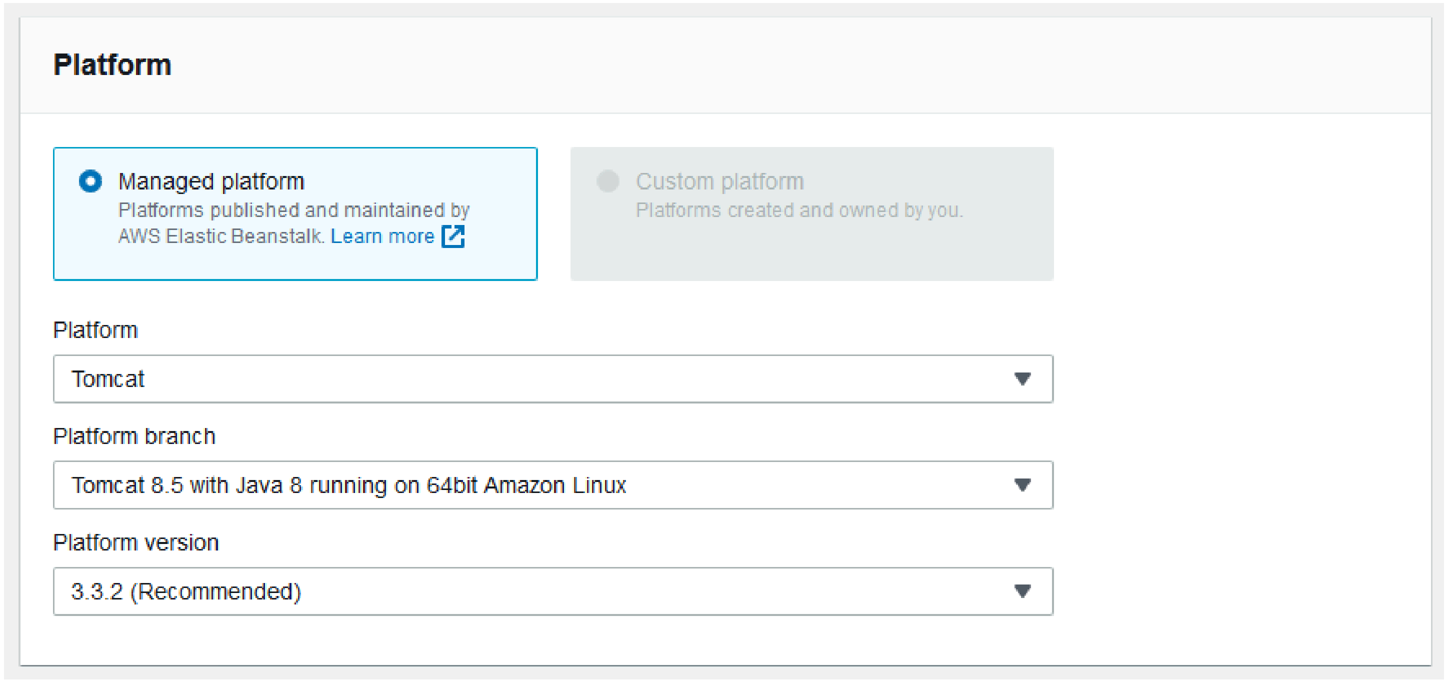
Select a platform for the new environment

You can create a new environment from two types of platforms:

- Managed platform
- Custom platform

Managed platform

In most cases you use an Elastic Beanstalk managed platform for your new environment. When the new environment wizard starts, it selects the **Managed platform** option by default.


Platform

● Managed platform
Platforms published and maintained by AWS Elastic Beanstalk. Learn more

○ Custom platform
Platforms created and owned by you.

Platform: Tomcat

Platform branch: Tomcat 8.5 with Java 8 running on 64bit Amazon Linux

Platform version: 3.3.2 (Recommended)

Select a platform, a platform branch within that platform, and a specific platform version in the branch. When you select a platform branch, the recommended version within the branch is selected by default. In addition, you can select any platform version you've used before.

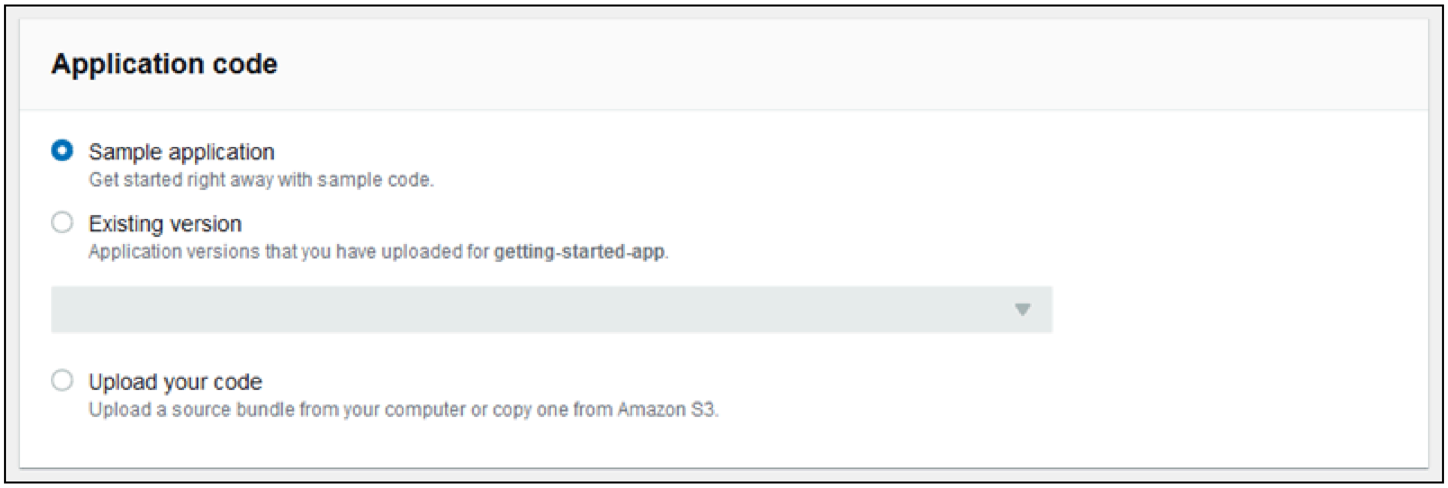
> **Note**
>
> For a production environment, we recommend that you choose a platform version in a supported platform branch. For details about platform branch states, see the *Platform Branch* definition in the the section called “Platforms glossary”.

Custom platform

If an off-the-shelf platform doesn't meet your needs, you can create a new environment from a custom platform. To specify a custom platform, choose the **Custom platform** option, and then select one of the available custom platforms. If there are no custom platforms available, this option is dimmed.

Provide application code

Now that you have selected the platform to use, the next step is to provide your application code.



Application code

[✓ Sample application] Get started right away with sample code.

Existing version Application versions that you have uploaded for **getting-started-app**.

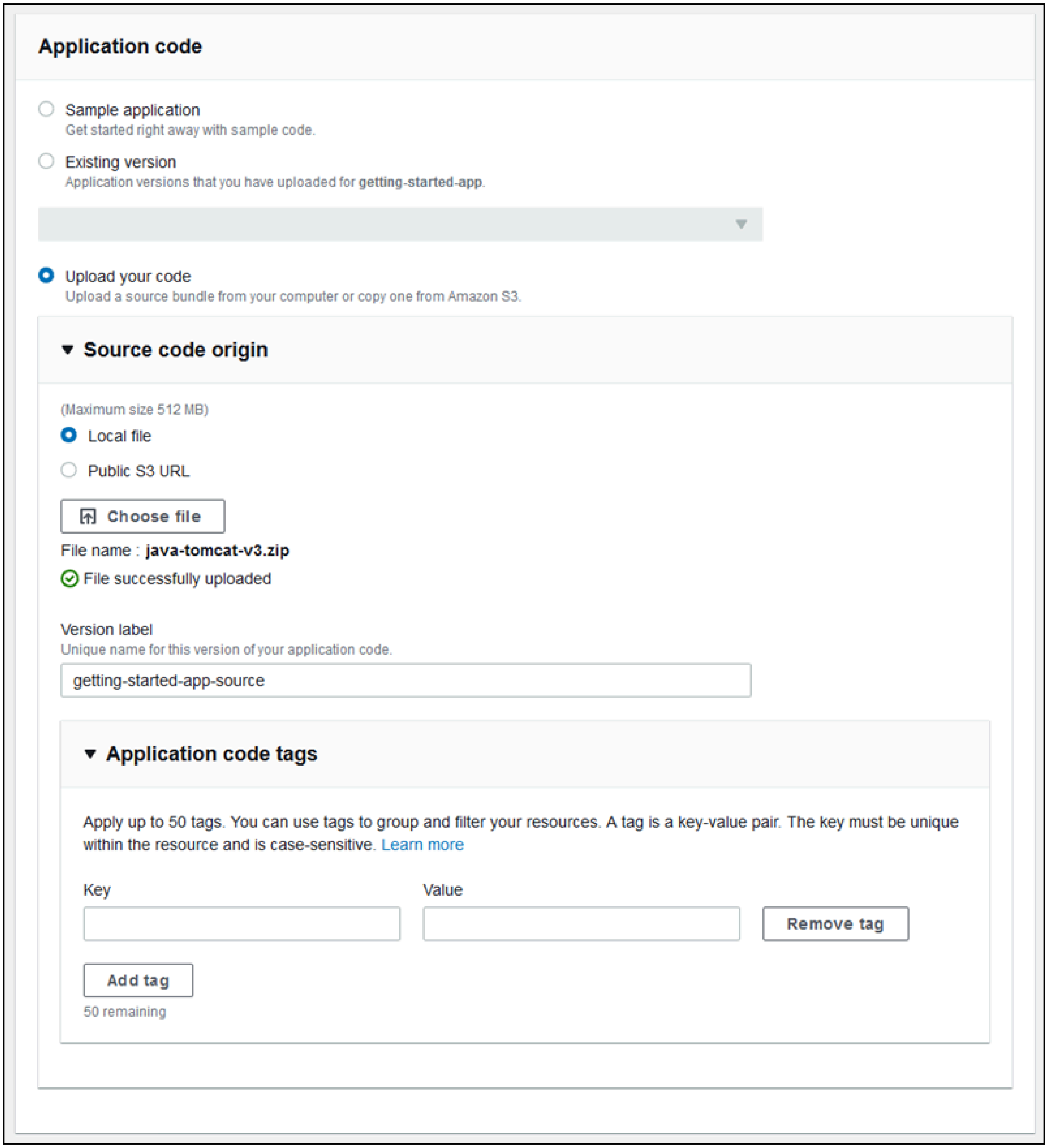
Upload your code Upload a source bundle from your computer or copy one from Amazon S3.

You have several options:

- You can use the sample application that Elastic Beanstalk provides for each platform.
- You can use code that you already deployed to Elastic Beanstalk. Choose **Existing version** and your application in the **Application code** section.
- You can upload new code. Choose **Upload your code**, and then choose **Upload**. You can upload new application code from a local file, or you can specify the URL for the Amazon S3 bucket that contains your application code.

> **Note**
>
> Depending on the platform version you selected, you can upload your application in a ZIP source bundle, a WAR file, or a plaintext Docker configuration. The file size limit is 500 MB.

When you choose to upload new code, you can also provide tags to associate with your new code. For more information about tagging an application version, see the section called “Tagging application versions”.


Application code

( ) Sample application
Get started right away with sample code.

( ) Existing version
Application versions that you have uploaded for **getting-started-app**.

(●) Upload your code
Upload a source bundle from your computer or copy one from Amazon S3.

Source code origin

(Maximum size 512 MB)

(●) Local file

( ) Public S3 URL

Choose file

File name : **java-tomcat-v3.zip**

✓ File successfully uploaded

Version label
Unique name for this version of your application code.

getting-started-app-source

Application code tags

Apply up to 50 tags. You can use tags to group and filter your resources. A tag is a key-value pair. The key must be unique within the resource and is case-sensitive. Learn more

Key | Value | Remove tag

Add tag

50 remaining

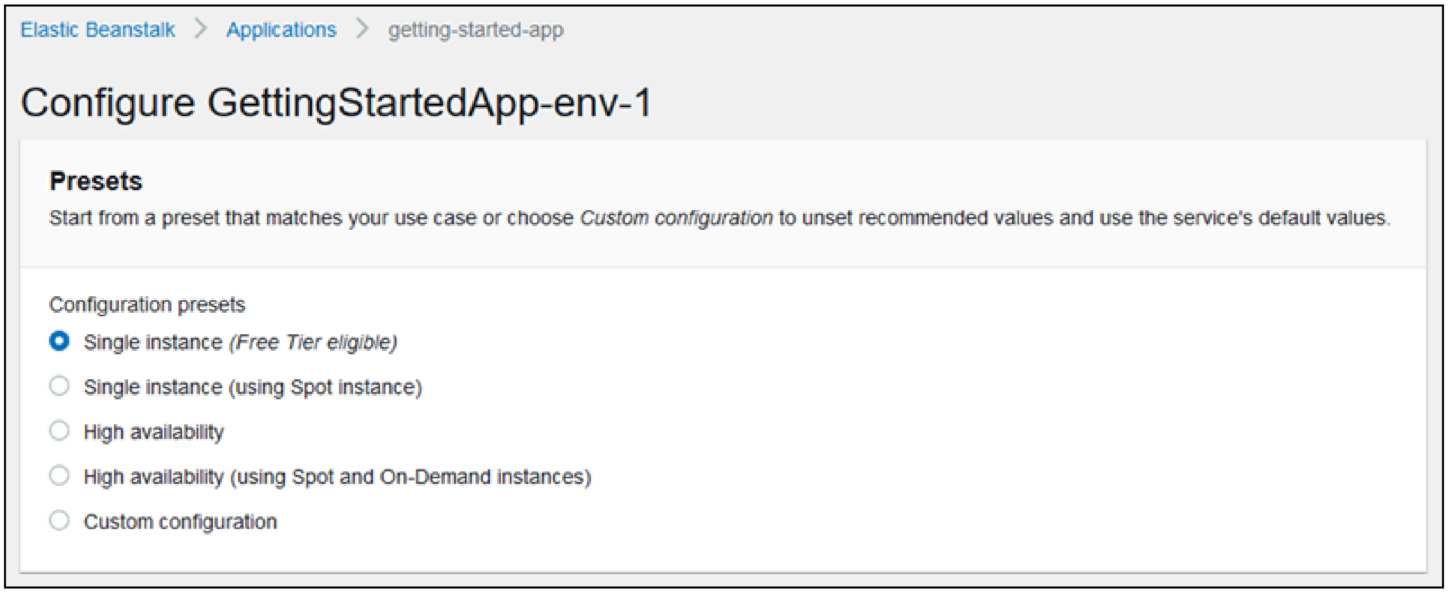
For quick environment creation using default configuration options, you can now choose **Create environment**. Choose **Configure more options** to make additional configuration changes, as described in the following sections.

Wizard configuration page

When you choose **Configure more options**, the wizard shows the **Configure** page. On this page you can select a configuration preset, change the platform version you want your environment to use, or make specific configuration choices for the new environment.

Choose a preset configuration

On the **Presets** section of the page, Elastic Beanstalk provides several configuration presets for different use cases. Each preset includes recommended values for several configuration options.



Elastic Beanstalk > Applications > getting-started-app

Configure GettingStartedApp-env-1

Presets

Start from a preset that matches your use case or choose *Custom configuration* to unset recommended values and use the service's default values.

Configuration presets

- [✓ Single instance *(Free Tier eligible)*]
- Single instance (using Spot instance)
- High availability
- High availability (using Spot and On-Demand instances)
- Custom configuration

The **High availability** presets include a load balancer, and are recommended for production environments. Choose them if you want an environment that can run multiple instances for high availability and scale in response to load. The **Single instance** presets are primarily recommended for development. Two of the presets enable Spot Instance requests. For details about Elastic Beanstalk capacity configuration, see Auto Scaling group.

The last preset, **Custom configuration**, removes all recommended values except role settings and uses the API defaults. Choose this option if you are deploying a source bundle with configuration files that set configuration options. **Custom configuration** is also selected automatically if you modify either the **Low cost** or **High availability** configuration presets.

Customize your configuration

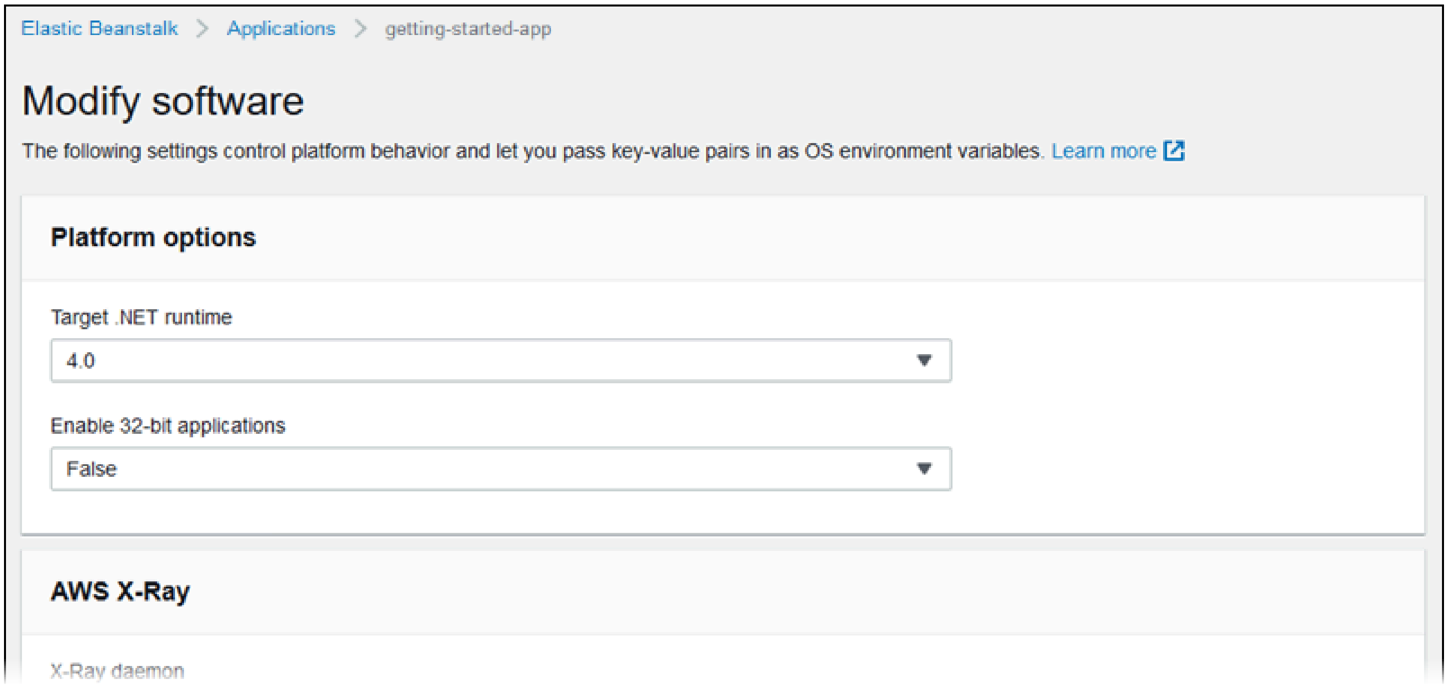
In addition to (or instead of) choosing a configuration preset, you can fine-tune configuration options in your environment. The **Configure** wizard wizard shows several configuration categories. Each configuration category displays a summary of values for a group of configuration settings. Choose **Edit** to edit this group of settings.

Configuration Categories

- Software settings
- Instances
- Capacity
- Load balancer
- Rolling updates and deployments
- Security
- Monitoring
- Managed updates
- Notifications
- Network
- Database
- Tags
- Worker environment

Software settings

Use the **Modify software** configuration page to configure the software on the Amazon Elastic Compute Cloud (Amazon EC2) instances that run your application. You can configure environment properties, AWS X-Ray debugging, instance log storing and streaming, and platform-specific settings. For details, see the section called “Environment variables and software settings”.


Elastic Beanstalk > Applications > getting-started-app

Modify software

The following settings control platform behavior and let you pass key-value pairs in as OS environment variables. Learn more

Platform options

Target .NET runtime: 4.0

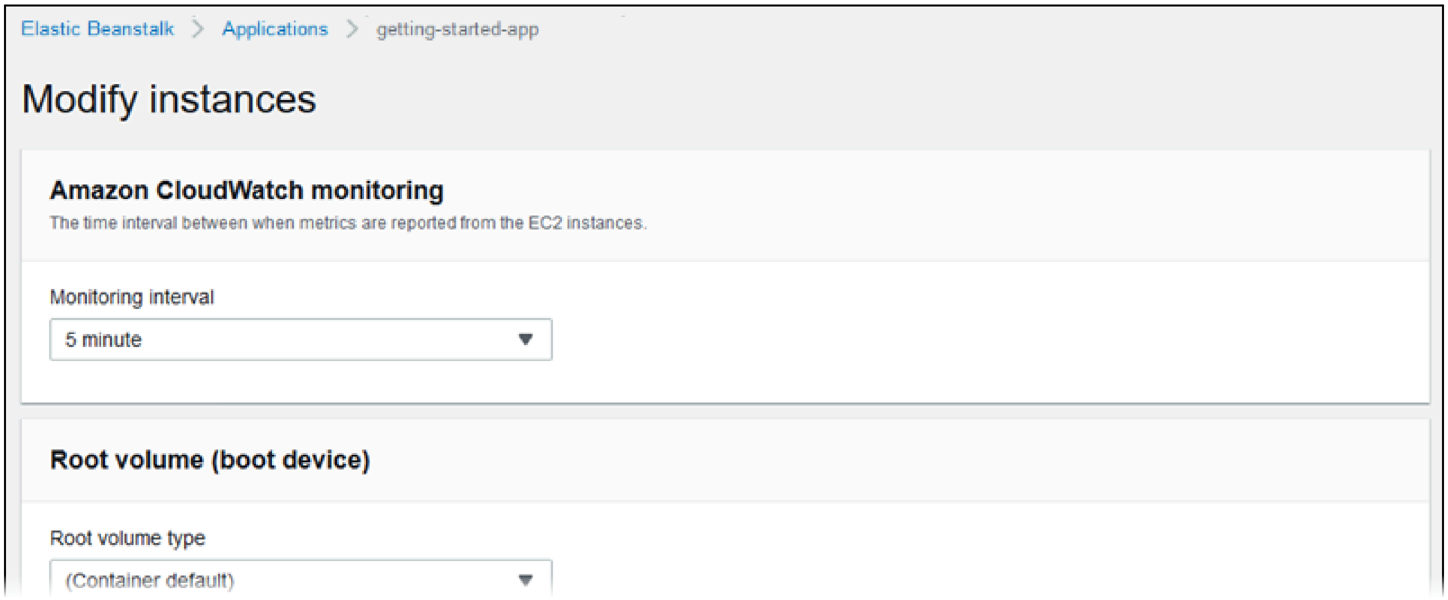
Enable 32-bit applications: False

AWS X-Ray

X-Ray daemon

Instances

Use the **Modify instances** configuration page to configure the Amazon EC2 instances that run your application. For details, see the section called "Amazon EC2 instances".


Elastic Beanstalk > Applications > getting-started-app

Modify instances

Amazon CloudWatch monitoring

The time interval between when metrics are reported from the EC2 instances.

Monitoring interval: 5 minute

Root volume (boot device)

Root volume type: (Container default)

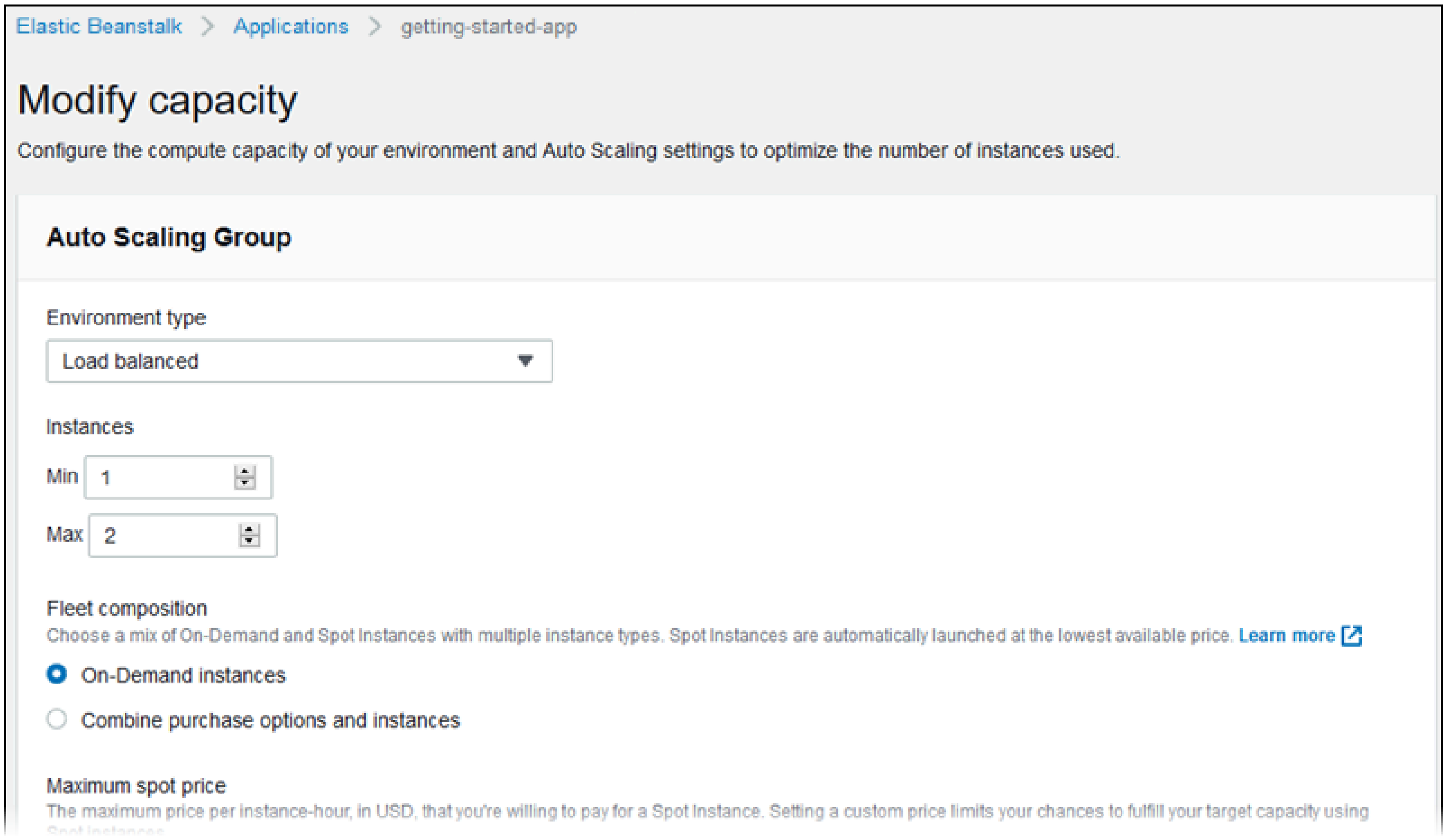
Capacity

Use the **Modify capacity** configuration page to configure the compute capacity of your environment and **Auto Scaling group** settings to optimize the number and type of instances you're using. You can also change your environment capacity based on triggers or on a schedule.

A load-balanced environment can run multiple instances for high availability and prevent downtime during configuration updates and deployments. In a load-balanced environment, the domain name maps to the load balancer. In a single-instance environment, it maps to an elastic IP address on the instance.

> **Warning**
>
> A single-instance environment isn't production ready. If the instance becomes unstable during deployment, or Elastic Beanstalk terminates and restarts the instance during a configuration update, your application can be unavailable for a period of time. Use single-instance environments for development, testing, or staging. Use load-balanced environments for production.

For more information about environment capacity settings, see the section called "Auto Scaling group" and the section called "Amazon EC2 instances".



Modify capacity

Configure the compute capacity of your environment and Auto Scaling settings to optimize the number of instances used.

Auto Scaling Group

Environment type: Load balanced

Instances

Min: 1

Max: 2

Fleet composition

Choose a mix of On-Demand and Spot Instances with multiple instance types. Spot Instances are automatically launched at the lowest available price. Learn more

[✓ On-Demand instances]

Combine purchase options and instances

Maximum spot price

The maximum price per instance-hour, in USD, that you're willing to pay for a Spot Instance. Setting a custom price limits your chances to fulfill your target capacity using

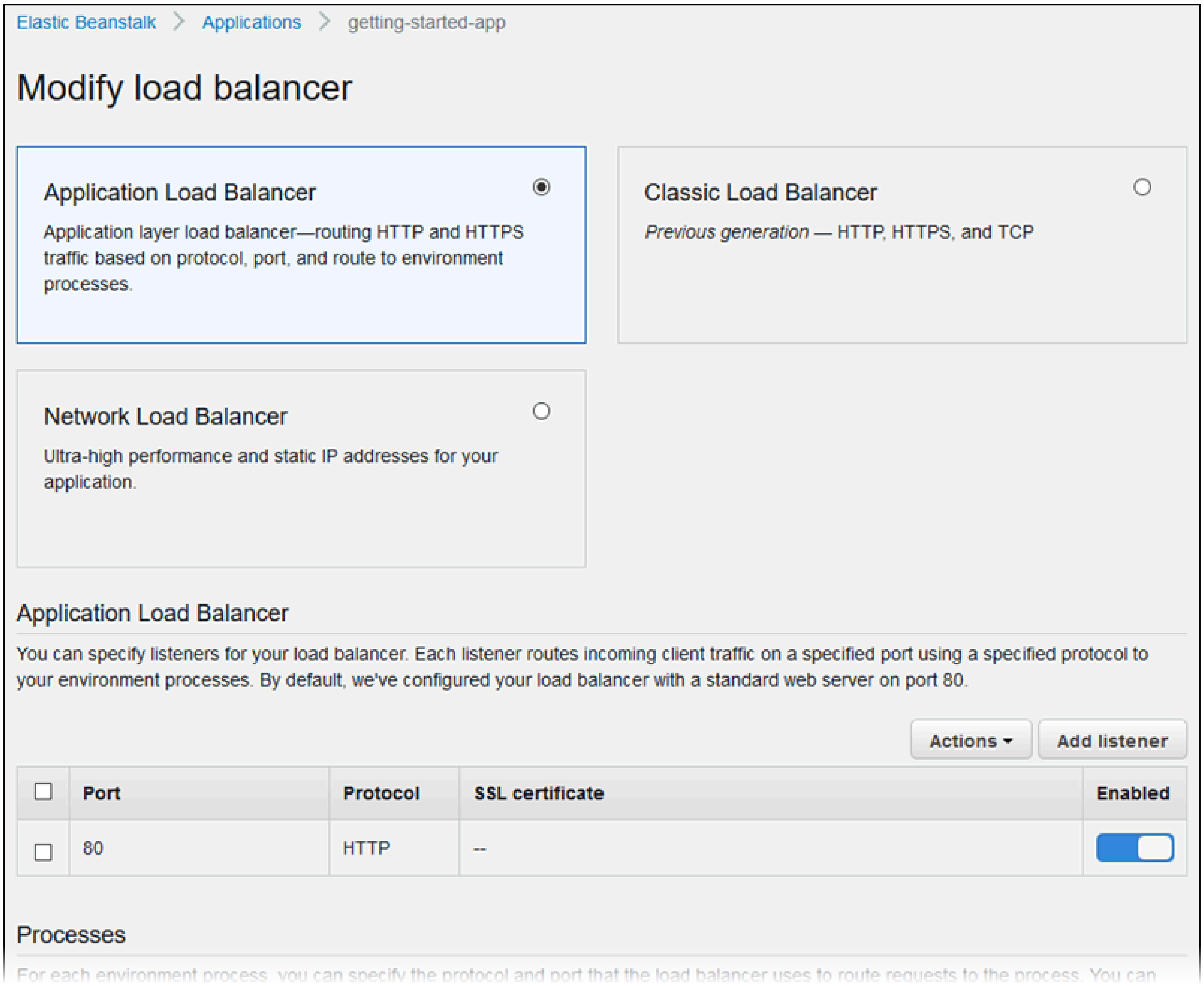
Load balancer

Use the **Modify load balancer** configuration page to select a load balancer type and to configure settings for it. In a load-balanced environment, your environment's load balancer is the entry point for all traffic headed for your application. Elastic Beanstalk supports several types of load balancer. By default, the Elastic Beanstalk console creates an Application Load Balancer and configures it to serve HTTP traffic on port 80.

Note

You can only select your environment's load balancer type during environment creation.

For more information about load balancer types and settings, see the section called “Load balancer” and the section called “HTTPS”.


Elastic Beanstalk > Applications > getting-started-app

Modify load balancer

[✓ Application Load Balancer] Application layer load balancer—routing HTTP and HTTPS traffic based on protocol, port, and route to environment processes.

Classic Load Balancer *Previous generation* — HTTP, HTTPS, and TCP

Network Load Balancer Ultra-high performance and static IP addresses for your application.

Application Load Balancer

You can specify listeners for your load balancer. Each listener routes incoming client traffic on a specified port using a specified protocol to your environment processes. By default, we've configured your load balancer with a standard web server on port 80.

	Port	Protocol	SSL certificate	Enabled
☐	80	HTTP	--	✓

Processes

For each environment process, you can specify the protocol and port that the load balancer uses to route requests to the process. You can

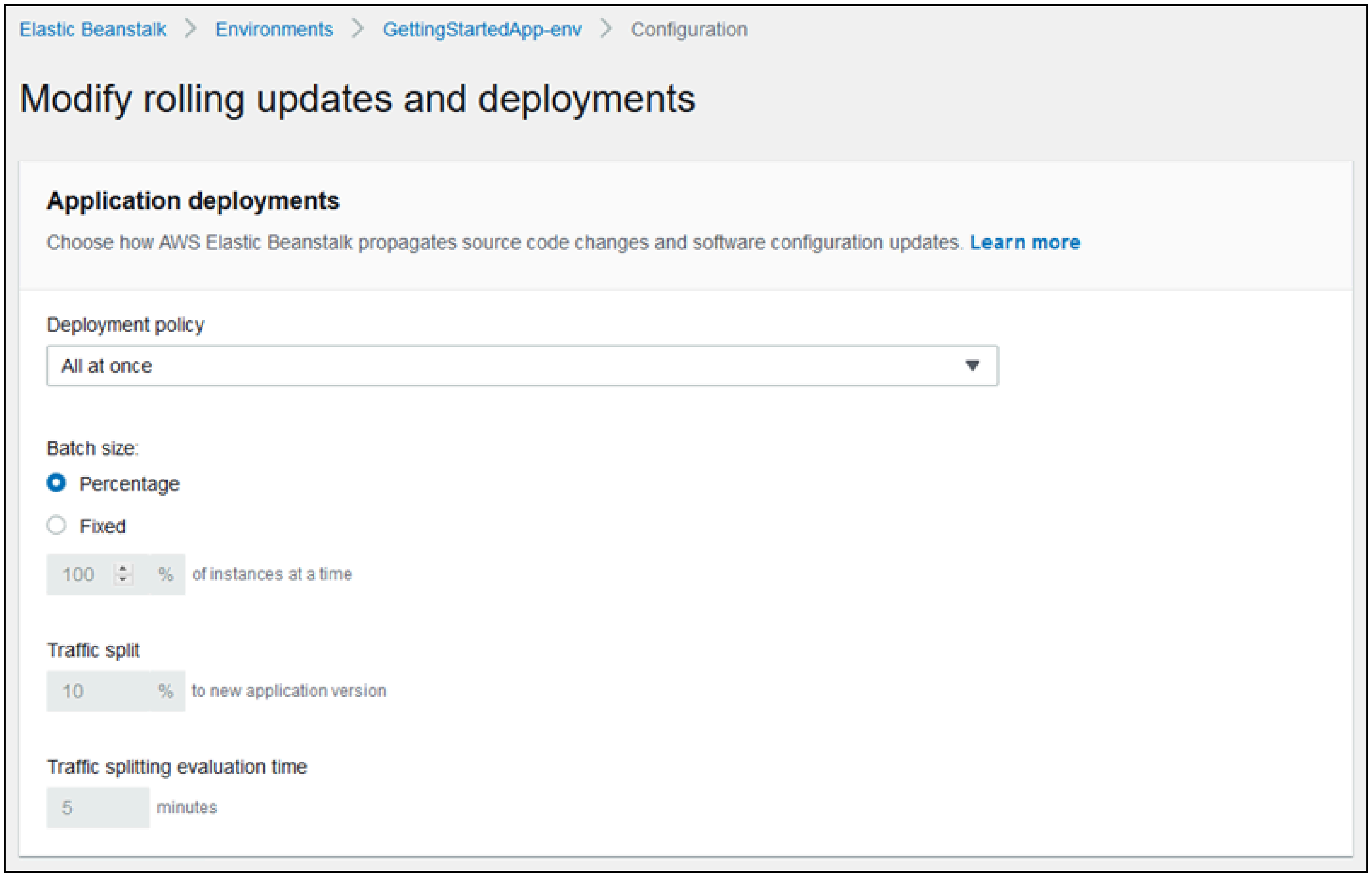
Note

The Classic Load Balancer (CLB) option is disabled on the **Create Environment** console wizard. If you have an existing environment configured with a Classic Load Balancer you can create a new one by cloning the existing environment using either the Elastic Beanstalk console or the EB CLI. You also have the option to use the EB CLI or the AWS CLI to create a new environment configured with a Classic Load Balancer. These command line tools will create a new environment with a CLB even if one doesn't already exist in your account.

Rolling updates and deployments

Use the **Modify rolling updates and deployments** configuration page to configure how Elastic Beanstalk processes application deployments and configuration updates for your environment.

Application deployments happen when you upload an updated application source bundle and deploy it to your environment. For more information about configuring deployments, see the section called "Deployment options".


Elastic Beanstalk > Environments > GettingStartedApp-env > Configuration

Modify rolling updates and deployments

Application deployments

Choose how AWS Elastic Beanstalk propagates source code changes and software configuration updates. Learn more

Deployment policy: All at once

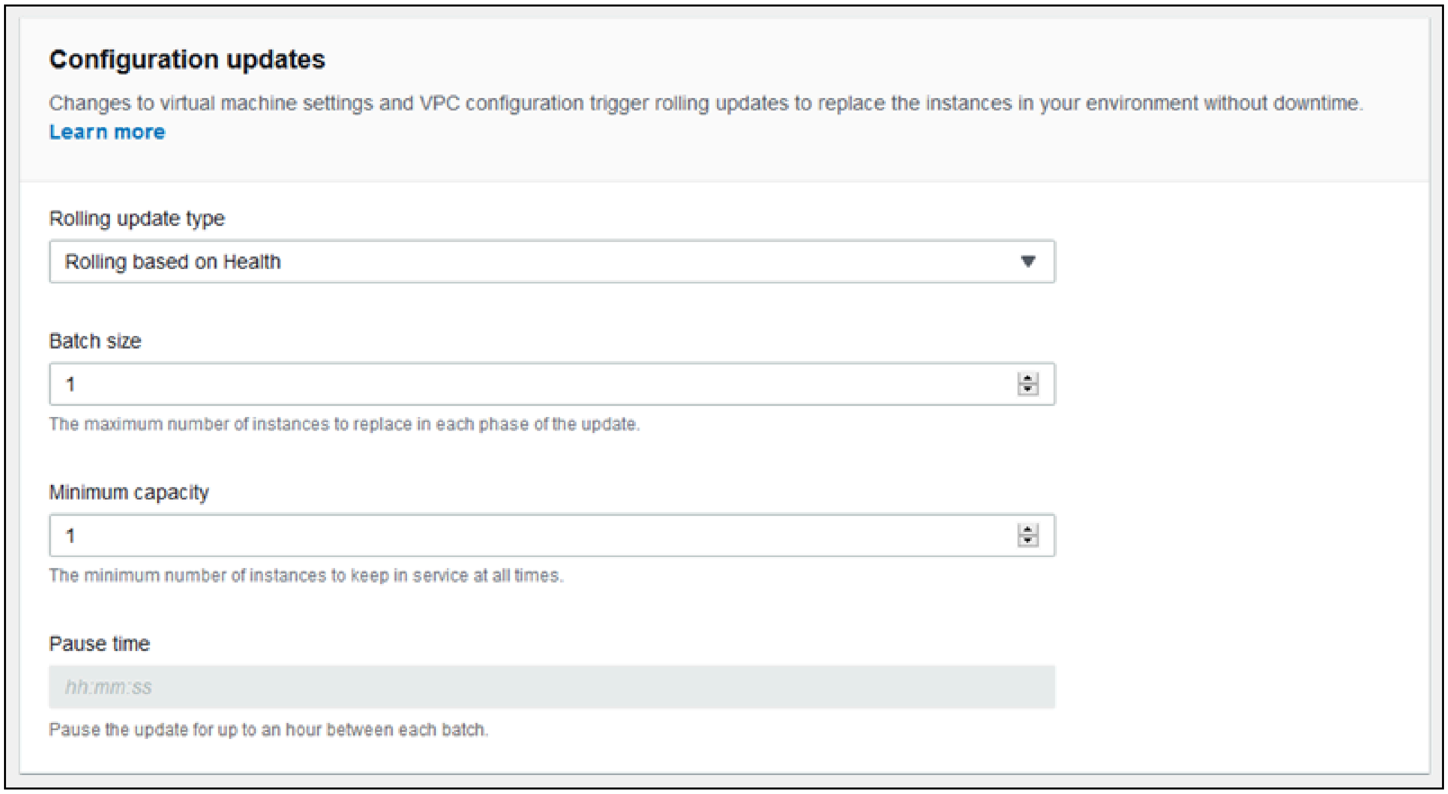
Batch size: [✓ Percentage] Fixed

100 % of instances at a time

Traffic split: 10 % to new application version

Traffic splitting evaluation time: 5 minutes

Configuration changes that modify the launch configuration or VPC settings require terminating all instances in your environment and replacing them. For more information about setting the update type and other options, see the section called "Configuration changes".



Configuration updates

Changes to virtual machine settings and VPC configuration trigger rolling updates to replace the instances in your environment without downtime.
Learn more

Rolling update type: Rolling based on Health

Batch size: 1

The maximum number of instances to replace in each phase of the update.

Minimum capacity: 1

The minimum number of instances to keep in service at all times.

Pause time: hh:mm:ss

Pause the update for up to an hour between each batch.

Security

Use the **Configure service access** page to configure service and instance security settings.

For a description of Elastic Beanstalk security concepts, see *Permissions*.

The first time you create an environment in the Elastic Beanstalk console, you must create an EC2 instance profile with a default set of permissions. If the **EC2 instance profile** dropdown list doesn't show any values to choose from, expand the procedure that follows. It provides steps to create a Role that you can subsequently select for the **EC2 instance profile**.

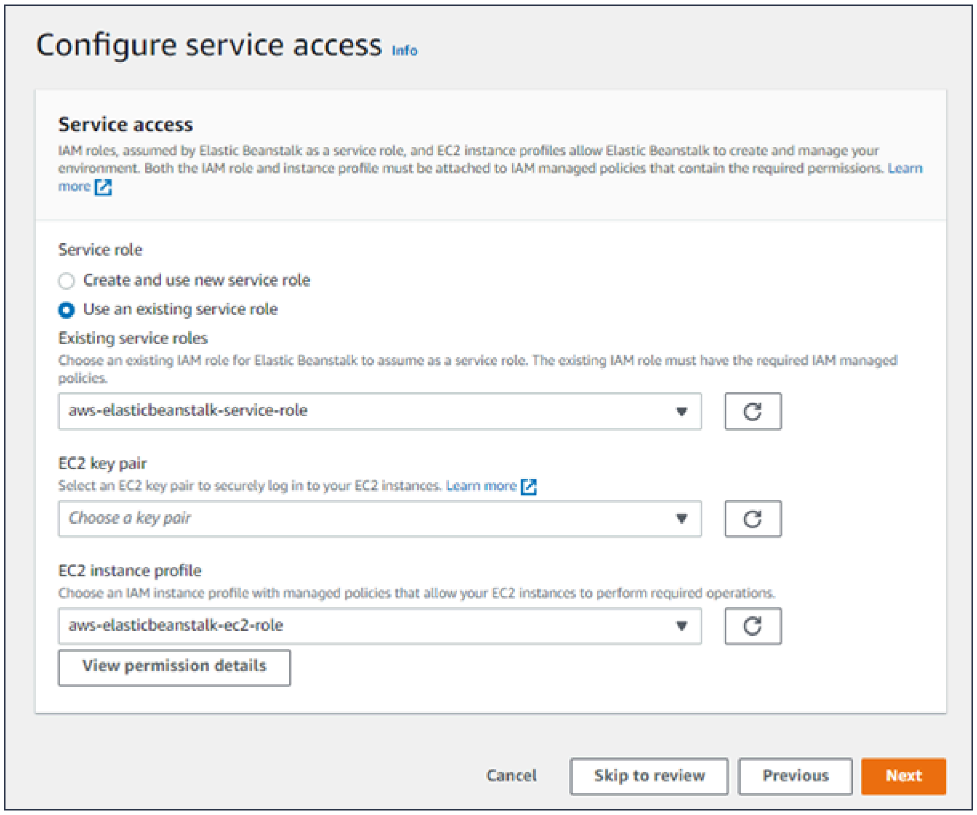
Create IAM Role for EC2 instance profile

To create the EC2 instance profile

1. Choose **Create role**.
2. For **Trusted entity type**, choose **AWS service**.
3. For **Use case**, choose **Elastic Beanstalk – Compute**.
4. Choose **Next**.
5. Verify that **Permissions policies** include the following, then choose **Next**:

  - `AWSElasticBeanstalkWebTier`
  - `AWSElasticBeanstalkWorkerTier`
  - `AWSElasticBeanstalkMulticontainerDocker`

6. Choose **Create role**.
7. Return to the **Configure service access** tab, refresh the list, then select the newly created EC2 instance profile.


Configure service access Info

Service access

IAM roles, assumed by Elastic Beanstalk as a service role, and EC2 instance profiles allow Elastic Beanstalk to create and manage your environment. Both the IAM role and instance profile must be attached to IAM managed policies that contain the required permissions. Learn more

Service role

○ Create and use new service role

● Use an existing service role

Existing service roles

Choose an existing IAM role for Elastic Beanstalk to assume as a service role. The existing IAM role must have the required IAM managed policies.

aws-elasticbeanstalk-service-role

EC2 key pair

Select an EC2 key pair to securely log in to your EC2 instances. Learn more

Choose a key pair

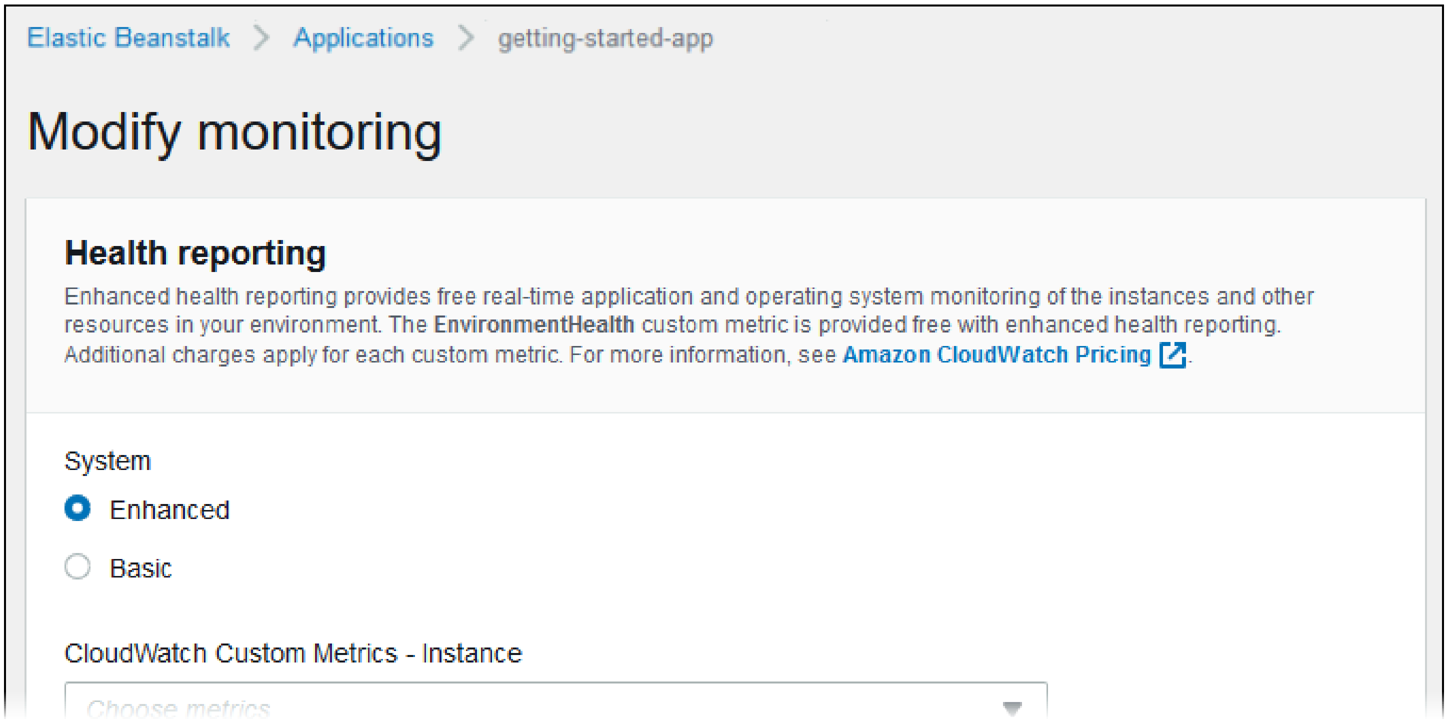
EC2 instance profile

Choose an IAM instance profile with managed policies that allow your EC2 instances to perform required operations.

aws-elasticbeanstalk-ec2-role

Monitoring

Use the **Modify monitoring** configuration page to configure health reporting, monitoring rules, and health event streaming. For details, see the section called “Enable enhanced health”, the section called “Enhanced health rules”, and the section called “Streaming environment health”.



Elastic Beanstalk > Applications > getting-started-app

Modify monitoring

Health reporting

Enhanced health reporting provides free real-time application and operating system monitoring of the instances and other resources in your environment. The **EnvironmentHealth** custom metric is provided free with enhanced health reporting. Additional charges apply for each custom metric. For more information, see **Amazon CloudWatch Pricing**.

System

[● Enhanced]

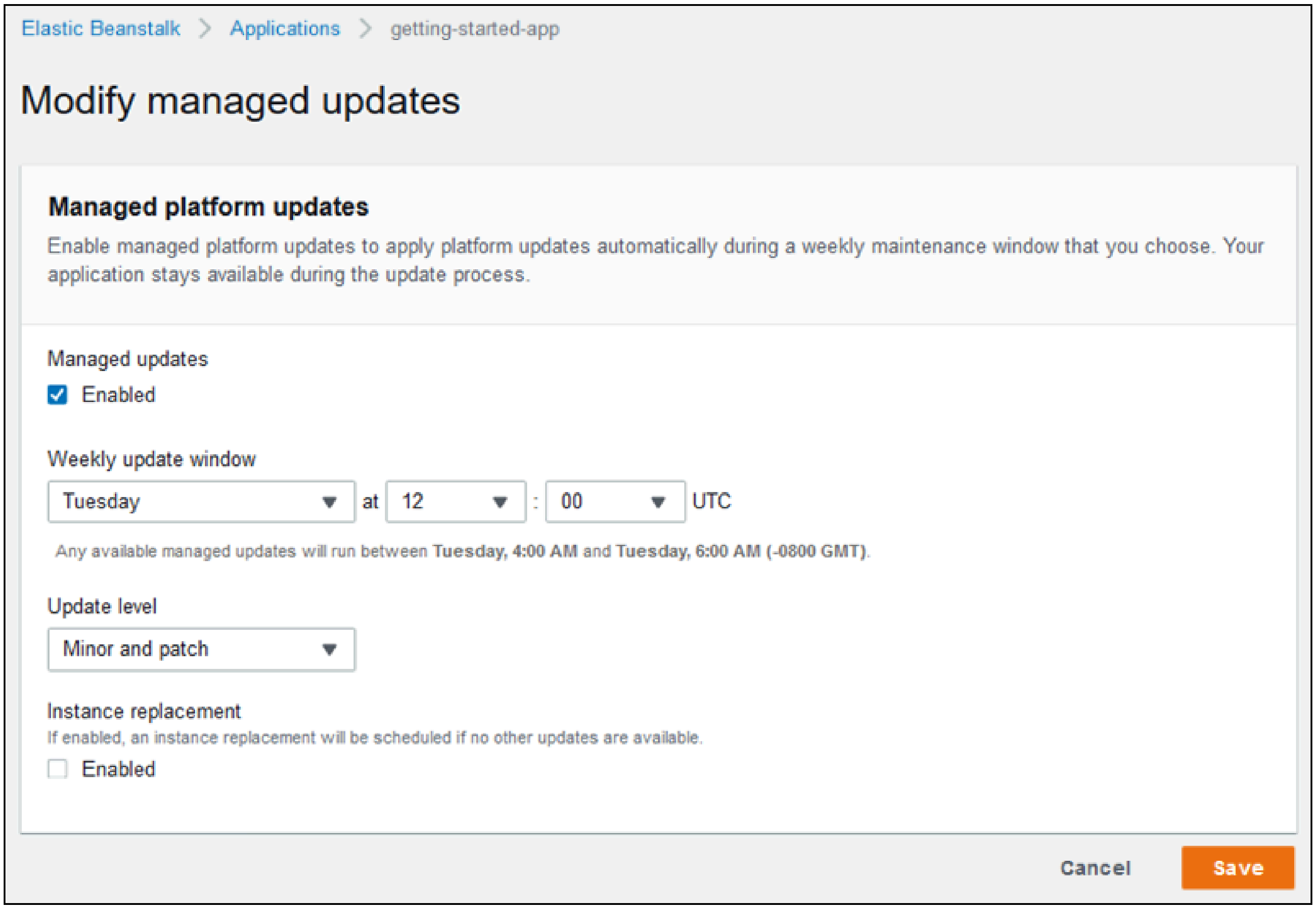
○ Basic

CloudWatch Custom Metrics - Instance

Choose metrics

Managed updates

Use the **Modify managed updates** configuration page to configure managed platform updates. You can decide if you want them enabled, set the schedule, and configure other properties. For details, see the section called "Managed updates".



Modify managed updates

Managed platform updates

Enable managed platform updates to apply platform updates automatically during a weekly maintenance window that you choose. Your application stays available during the update process.

Managed updates

[✓] Enabled

Weekly update window

Tuesday at 12 : 00 UTC

Any available managed updates will run between **Tuesday, 4:00 AM** and **Tuesday, 6:00 AM (-0800 GMT)**.

Update level

Minor and patch

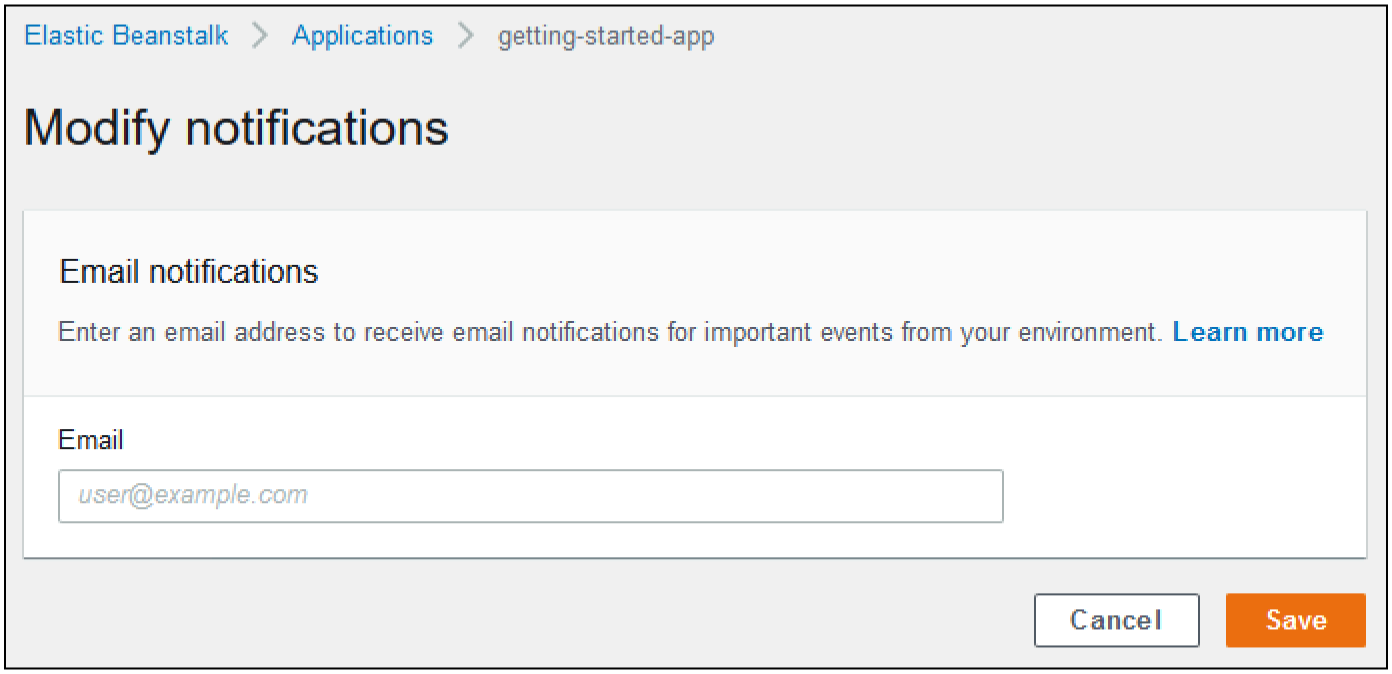
Instance replacement

If enabled, an instance replacement will be scheduled if no other updates are available.

[ ] Enabled

Notifications

Use the **Modify notifications** configuration page to specify an email address to receive email notifications for important events from your environment.


Elastic Beanstalk > Applications > getting-started-app

Modify notifications

Email notifications

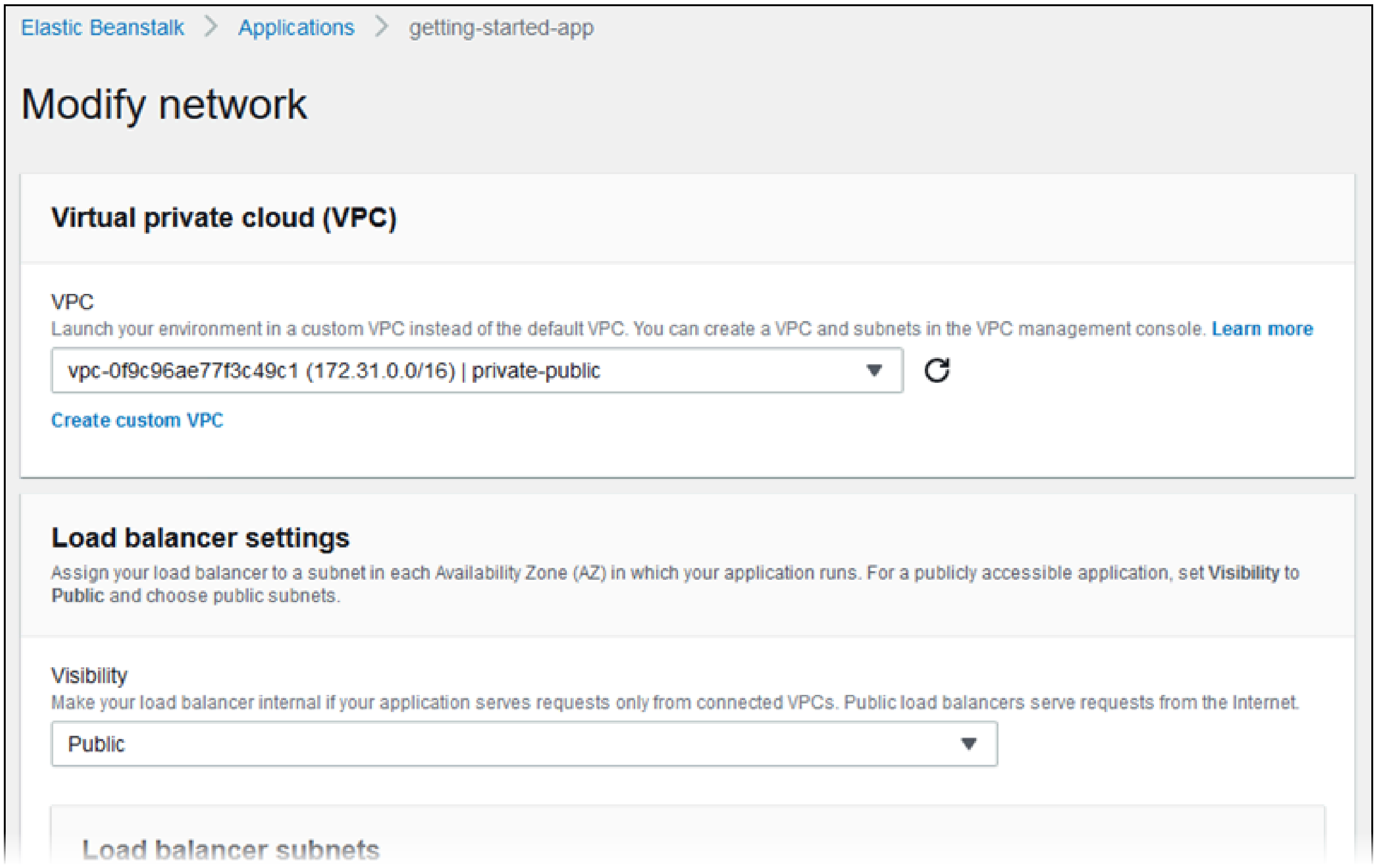
Enter an email address to receive email notifications for important events from your environment. Learn more

Email

user@example.com

Network

If you have created a [custom VPC](), the **Modify network** configuration page to configure your environment to use it. If you don't choose a VPC, Elastic Beanstalk uses the default VPC and subnets.



Elastic Beanstalk > Applications > getting-started-app

Modify network

Virtual private cloud (VPC)

VPC
Launch your environment in a custom VPC instead of the default VPC. You can create a VPC and subnets in the VPC management console. Learn more

vpc-0f9c96ae77f3c49c1 (172.31.0.0/16) | private-public

Create custom VPC

Load balancer settings
Assign your load balancer to a subnet in each Availability Zone (AZ) in which your application runs. For a publicly accessible application, set **Visibility** to **Public** and choose public subnets.

Visibility
Make your load balancer internal if your application serves requests only from connected VPCs. Public load balancers serve requests from the Internet.

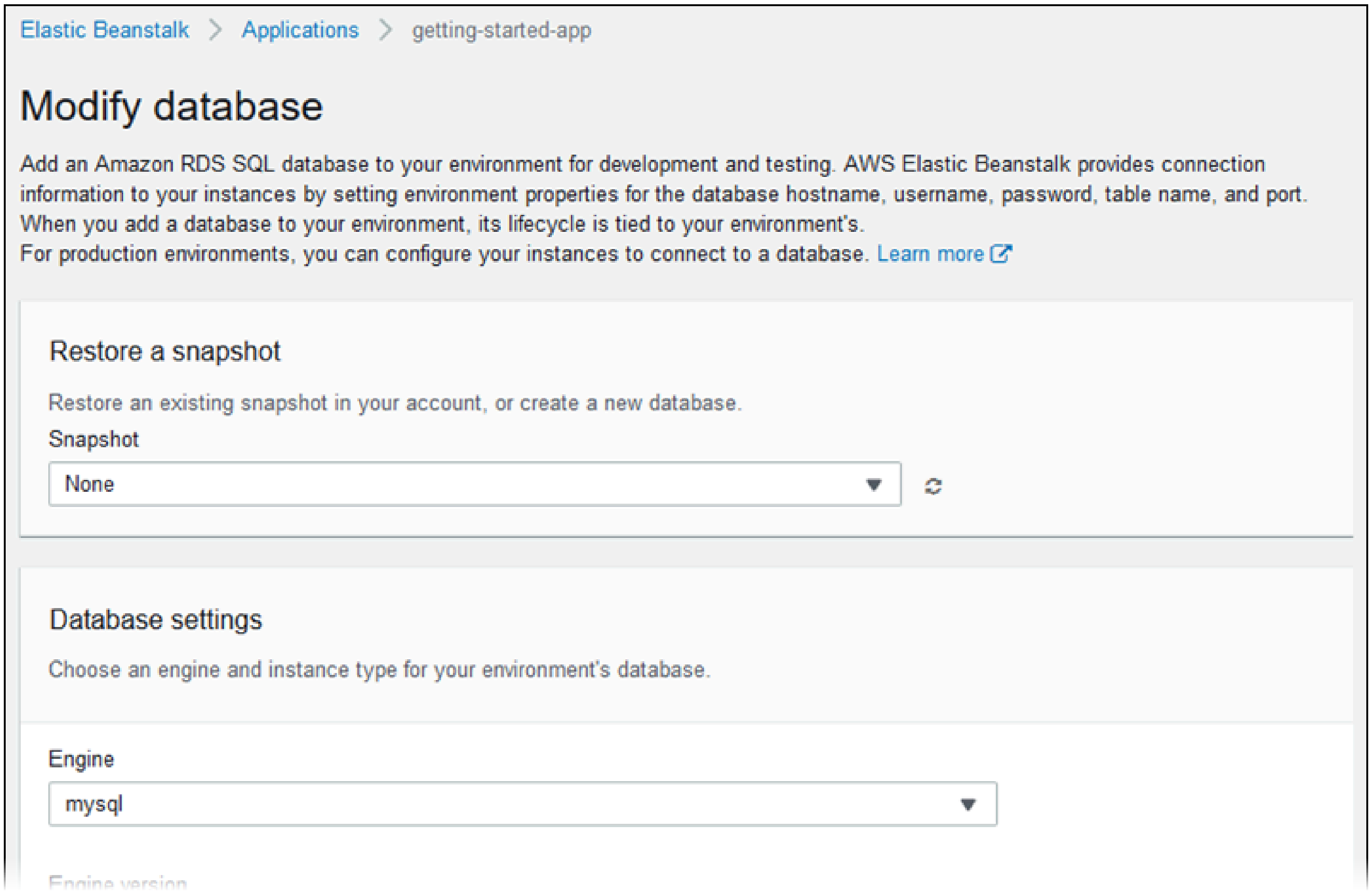
Public

Load balancer subnets

Database

Use the **Modify database** configuration page to add an Amazon Relational Database Service (Amazon RDS) database to your environment for development and testing. Elastic Beanstalk provides connection information to your instances by setting environment properties for the database hostname, user name, password, table name, and port.

For details, see the section called "Database".



Elastic Beanstalk > Applications > getting-started-app

Modify database

Add an Amazon RDS SQL database to your environment for development and testing. AWS Elastic Beanstalk provides connection information to your instances by setting environment properties for the database hostname, username, password, table name, and port. When you add a database to your environment, its lifecycle is tied to your environment's.
For production environments, you can configure your instances to connect to a database. Learn more

Restore a snapshot

Restore an existing snapshot in your account, or create a new database.

Snapshot: None

Database settings

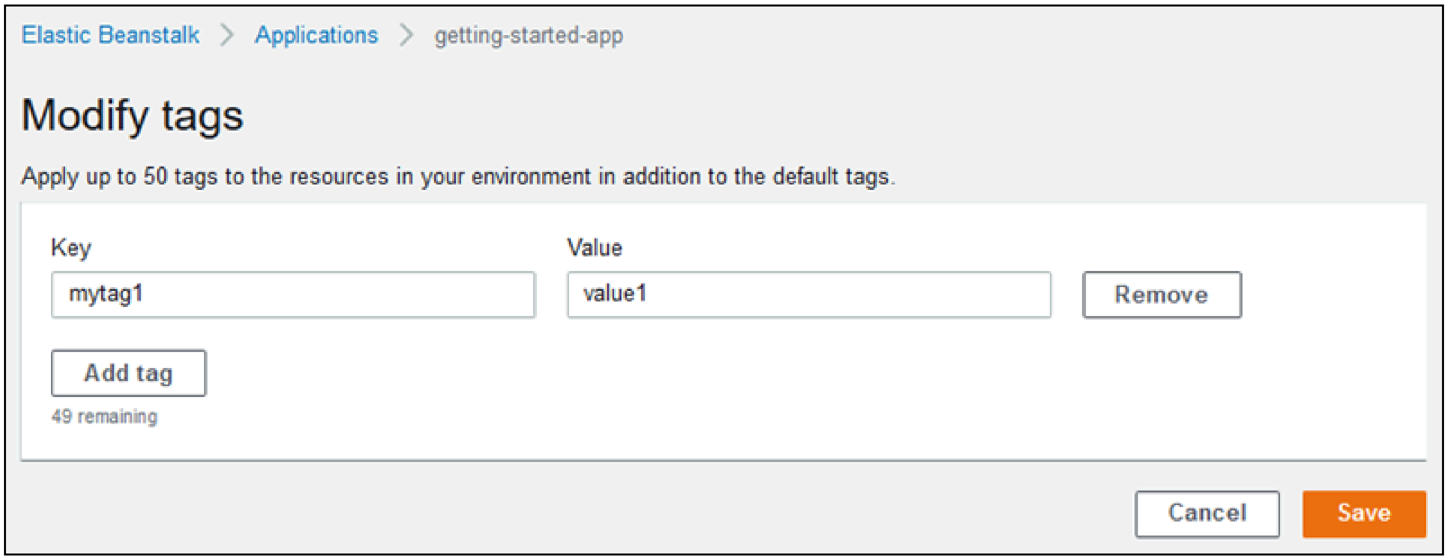
Choose an engine and instance type for your environment's database.

Engine: mysql

Engine version

Tags

Use the **Modify tags** configuration page to add tags to the resources in your environment. For more information about environment tagging, see Tagging resources in your Elastic Beanstalk environments.


Elastic Beanstalk > Applications > getting-started-app

Modify tags

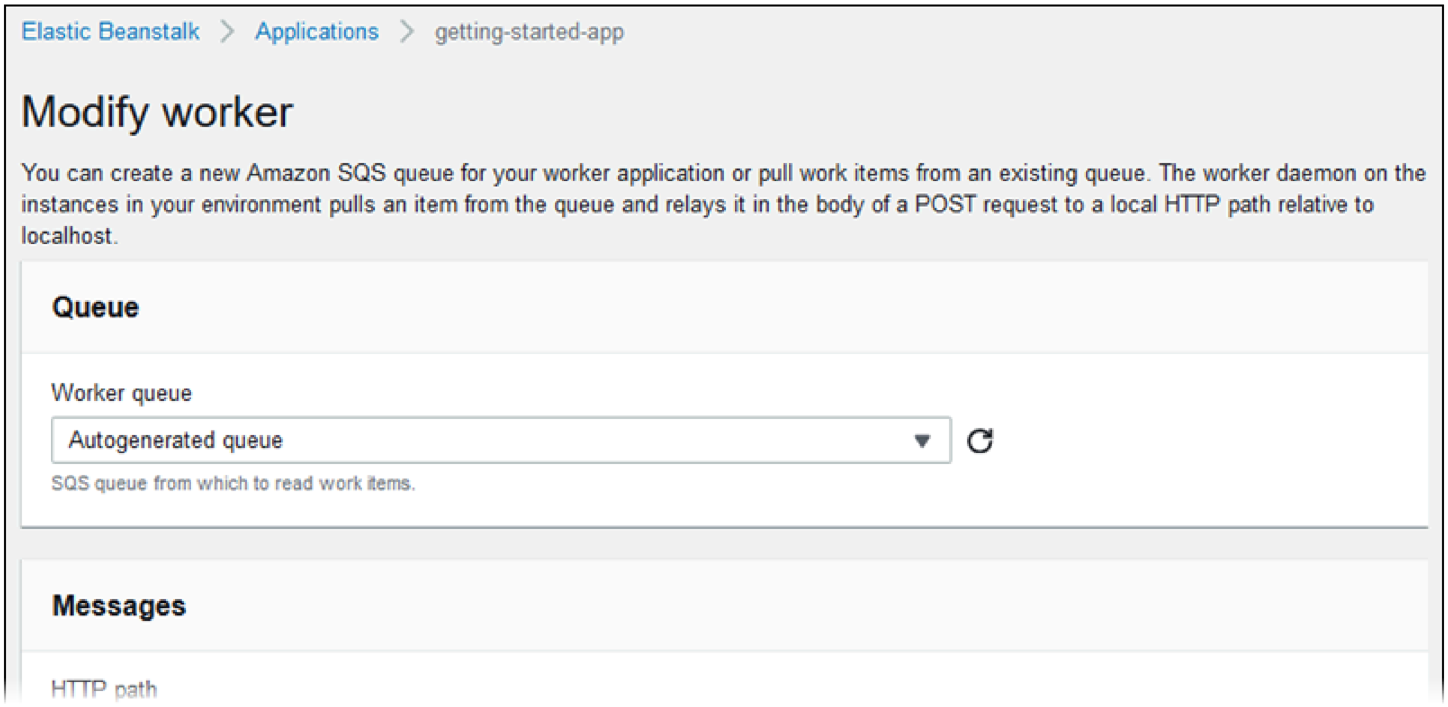
Apply up to 50 tags to the resources in your environment in addition to the default tags.

Key	Value
mytag1	value1

Worker environment

If you're creating a *worker tier environment*, use the **Modify worker** configuration page to configure the worker environment. The worker daemon on the instances in your environment pulls items from an Amazon Simple Queue Service (Amazon SQS) queue and relays them as post messages to your worker application. You can choose the Amazon SQS queue that the worker daemon reads from (auto-generated or existing). You can also configure the messages that the worker daemon sends to your application.

For more information, see the section called "Worker environments".


Elastic Beanstalk > Applications > getting-started-app

Modify worker

You can create a new Amazon SQS queue for your worker application or pull work items from an existing queue. The worker daemon on the instances in your environment pulls an item from the queue and relays it in the body of a POST request to a local HTTP path relative to localhost.

Queue

Worker queue: Autogenerated queue

SQS queue from which to read work items.

Messages

HTTP path

Clone an Elastic Beanstalk environment

You can use an existing Elastic Beanstalk environment as the basis for a new environment by cloning the existing environment. For example, you might want to create a clone so that you can use a newer version of the platform branch used by the original environment's platform. Elastic Beanstalk configures the clone with the environment settings used by the original environment. By cloning an existing environment instead of creating a new environment, you don't have to manually configure option settings, environment variables, and other settings that you made with the Elastic Beanstalk service. Elastic Beanstalk also creates a copy of any AWS resource associated with the original environment.

It's important to be aware of the following situations:

- During the cloning process, Elastic Beanstalk doesn't copy data from Amazon RDS to the clone.
- Elastic Beanstalk doesn't include any unmanaged changes to resources in the clone. Changes to AWS resources that you make using tools other than the Elastic Beanstalk console, command-line tools, or API are considered unmanaged changes.
- The security groups for ingress are considered unmanaged changes. Cloned Elastic Beanstalk environments do not carry over the security groups for ingress, leaving the environment open to all internet traffic. You'll need to reestablish ingress security groups for the cloned environment.

You can only clone an environment to a different platform version of the same platform branch. A different platform branch isn't guaranteed to be compatible. To use a different platform branch, you have to manually create a new environment, deploy your application code, and make any necessary changes in code and options to ensure your application works correctly on the new platform branch.

AWS management console

Important

Cloned Elastic Beanstalk environments do not carry over the security groups for ingress, leaving the environment open to all internet traffic. You'll need to reestablish ingress security groups for the cloned environment.
You can see resources that may not be cloned by checking the drift status of your environment configuration. For more information, see Detect drift on an entire CloudFormation stack in the *AWS CloudFormation User Guide*.

To clone an environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the environment overview page, choose **Actions**.
4. Choose **Clone environment**.
5. On the **Clone environment** page, review the information in the **Original Environment** section to verify that you chose the environment from which you want to create a clone.
6. In the **New Environment** section, you can optionally change the **Environment name**, **Environment URL**, **Description**, **Platform version**, and **Service role** values that Elastic Beanstalk automatically set based on the original environment.

   Note

   If the platform version used in the original environment isn't the one recommended for use in the platform branch, you are warned that a different platform version is recommended. Choose **Platform version**, and you can see the recommended platform version on the list—for example, **3.3.2 (Recommended)**.
7. When you are ready, choose **Clone**.

Elastic Beanstalk command line interface (EB CLI)

Important

Cloned Elastic Beanstalk environments do not carry over the security groups for ingress, leaving the environment open to all internet traffic. You'll need to reestablish ingress security groups for the cloned environment.
You can see resources that may not be cloned by checking the drift status of your environment configuration. For more information, see Detect drift on an entire CloudFormation stack in the *AWS CloudFormation User Guide*.

Use the **eb clone** command to clone a running environment, as follows.

```
~/workspace/my-app$ eb clone my-env1
```

```
Enter name for Environment Clone
(default is my-env1-clone): my-env2
Enter DNS CNAME prefix
(default is my-env1-clone): my-env2
```

You can specify the name of the source environment in the clone command, or leave it out to clone the default environment for the current project folder. The EB CLI prompts you to enter a name and DNS prefix for the new environment.

By default, **eb clone** creates the new environment with the latest available version of the source environment's platform. To force the EB CLI to use the same version, even if there is a newer version available, use the `--exact` option.

```
~/workspace/my-app$ eb clone --exact
```

For more information about this command, see eb clone.

Terminate an Elastic Beanstalk environment

You can terminate a running AWS Elastic Beanstalk environment using the Elastic Beanstalk console. By doing this, you avoid incurring charges for unused AWS resources.

Note

You can always launch a new environment using the same version later.

If you have data from an environment that you want to preserve, set the database deletion policy to `Retain` before terminating the environment. This keeps the database operational outside of Elastic Beanstalk. After this, any Elastic Beanstalk environments must connect to it as an external database. If you want to back up the data without keeping the database operational, set the deletion policy to take a snapshot of the database before terminating the environment. For more information, see Database lifecycle in the *Configuring environments* chapter of this guide.

Elastic Beanstalk might fail to terminate your environment. One common reason is that the security group of another environment has a dependency on the security group of the environment that you want to terminate. For instructions on how to avoid this problem, see EC2 security groups on the *EC2 Instances* page of this guide.

Important

If you terminate an environment, you must also delete any CNAME mappings that you created, as other customers can reuse an available hostname. Be sure to delete DNS records that point to your terminated environment to prevent a *dangling DNS entry*. A dangling DNS entry can expose internet traffic destined for your domain to security vulnerabilities. It can also present other risks.

For more information, see Protection from dangling delegation records in Route 53 in the *Amazon Route 53 Developer Guide*. You can also learn more about dangling DNS entries in Enhanced Domain Protections for Amazon CloudFront Requests in the *AWS Security Blog*.

Elastic Beanstalk console

To terminate an environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

   Note

   When you terminate your environment, the CNAME that's associated with the terminated environment is freed up to be used by anyone.

   It takes a few minutes for Elastic Beanstalk to terminate the AWS resources that are running in the environment.

AWS CLI

To terminate an environment

- Run the following command.

```
$ aws elasticbeanstalk terminate-environment --environment-name my-env
```

API

To terminate an environment

- Call `TerminateEnvironment` with the following parameter:

  `EnvironmentName` = `SampleAppEnv`

```
https://elasticbeanstalk.us-west-2.amazon.com/?EnvironmentName=SampleAppEnv
&Operation=TerminateEnvironment
&AuthParams
```

Creating Elastic Beanstalk environments with the AWS CLI

For details about the AWS CLI commands for Elastic Beanstalk, see the AWS CLI Command Reference.

1. Check if the CNAME for the environment is available.

```
$ aws elasticbeanstalk check-dns-availability --cname-prefix my-cname
{
    "Available": true,
    "FullyQualifiedCNAME": "my-cname.elasticbeanstalk.com"
}
```

2. Make sure your application version exists.

```
$ aws elasticbeanstalk describe-application-versions --application-name my-app --
version-label v1
```

   If you don't have an application version for your source yet, create it. For example, the following command creates an application version from a source bundle in Amazon Simple Storage Service (Amazon S3).

```
$ aws elasticbeanstalk create-application-version --application-name my-app --
version-label v1 --source-bundle S3Bucket=amzn-s3-demo-bucket,S3Key=my-source-
bundle.zip
```

3. Create a configuration template for the application.

```
$ aws elasticbeanstalk create-configuration-template --application-name my-app --
template-name v1 --solution-stack-name "64bit Amazon Linux 2015.03 v2.0.0 running
 Ruby 2.2 (Passenger Standalone)"
```

4. Create environment.

```
$ aws elasticbeanstalk create-environment --cname-prefix my-cname --application-
name my-app --template-name v1 --version-label v1 --environment-name v1clone --
option-settings file://options.txt
```

   Option Settings are defined in the **options.txt** file:

```
[
    {
        "Namespace": "aws:autoscaling:launchconfiguration",
        "OptionName": "IamInstanceProfile",
        "Value": "aws-elasticbeanstalk-ec2-role"
    }
]
```

   The above option setting defines the IAM instance profile. You can specify the ARN or the profile name.

5. Determine if the new environment is Green and Ready.

```
$ aws elasticbeanstalk describe-environments --environment-names my-env
```

   If the new environment does not come up Green and Ready, you should decide if you want to retry the operation or leave the environment in its current state for investigation. Make sure to terminate the environment after you are finished, and clean up any unused resources.

Note

You can adjust the timeout period if the environment doesn't launch in a reasonable time.

Creating Elastic Beanstalk environments with the API

1. Call `CheckDNSAvailability` with the following parameter:

   - `CNAMEPrefix` = `SampleApp`

   Example

   ```
   https://elasticbeanstalk.us-east-2.amazonaws.com/?CNAMEPrefix=sampleapplication
   &Operation=CheckDNSAvailability
   &AuthParams
   ```

2. Call `DescribeApplicationVersions` with the following parameters:

   - `ApplicationName` = `SampleApp`
   - `VersionLabel` = `Version2`

   Example

   ```
   https://elasticbeanstalk.us-east-2.amazonaws.com/?ApplicationName=SampleApp
   &VersionLabel=Version2
   &Operation=DescribeApplicationVersions
   &AuthParams
   ```

3. Call `CreateConfigurationTemplate` with the following parameters:

   - `ApplicationName` = `SampleApp`
   - `TemplateName` = `MyConfigTemplate`
   - `SolutionStackName` = `64bit%20Amazon%20Linux%202015.03%20v2.0.0%20running%20Ruby%202.2%20(Passenger%20Standalone)`

Example

```
https://elasticbeanstalk.us-east-2.amazonaws.com/?ApplicationName=SampleApp
&TemplateName=MyConfigTemplate
&Operation=CreateConfigurationTemplate
&SolutionStackName=64bit%20Amazon%20Linux%202015.03%20v2.0.0%20running%20Ruby
%202.2%20(Passenger%20Standalone)
&AuthParams
```

4. Call `CreateEnvironment` with one of the following sets of parameters.

   a. Use the following for a web server environment tier:

      - `EnvironmentName` = `SampleAppEnv2`
      - `VersionLabel` = `Version2`
      - `Description` = `description`
      - `TemplateName` = `MyConfigTemplate`
      - `ApplicationName` = `SampleApp`
      - `CNAMEPrefix` = `sampleapplication`
      - `OptionSettings.member.1.Namespace` = `aws:autoscaling:launchconfiguration`
      - `OptionSettings.member.1.OptionName` = `IamInstanceProfile`
      - `OptionSettings.member.1.Value` = `aws-elasticbeanstalk-ec2-role`

      Example

```
https://elasticbeanstalk.us-east-2.amazonaws.com/?ApplicationName=SampleApp
&VersionLabel=Version2
&EnvironmentName=SampleAppEnv2
&TemplateName=MyConfigTemplate
&CNAMEPrefix=sampleapplication
&Description=description
&Operation=CreateEnvironment
&OptionSettings.member.1.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.1.OptionName=IamInstanceProfile
&OptionSettings.member.1.Value=aws-elasticbeanstalk-ec2-role
```

```
&AuthParams
```

b. Use the following for a worker environment tier:

- `EnvironmentName` = `SampleAppEnv2`
- `VersionLabel` = `Version2`
- `Description` = `description`
- `TemplateName` = `MyConfigTemplate`
- `ApplicationName` = `SampleApp`
- `Tier` = `Worker`
- `OptionSettings.member.1.Namespace` = `aws:autoscaling:launchconfiguration`
- `OptionSettings.member.1.OptionName` = `IamInstanceProfile`
- `OptionSettings.member.1.Value` = `aws-elasticbeanstalk-ec2-role`
- `OptionSettings.member.2.Namespace` = `aws:elasticbeanstalk:sqsd`
- `OptionSettings.member.2.OptionName` = `WorkerQueueURL`
- `OptionSettings.member.2.Value` = `sqsd.elasticbeanstalk.us-east-2.amazonaws.com`
- `OptionSettings.member.3.Namespace` = `aws:elasticbeanstalk:sqsd`
- `OptionSettings.member.3.OptionName` = `HttpPath`
- `OptionSettings.member.3.Value` = `/`
- `OptionSettings.member.4.Namespace` = `aws:elasticbeanstalk:sqsd`
- `OptionSettings.member.4.OptionName` = `MimeType`
- `OptionSettings.member.4.Value` = `application/json`
- `OptionSettings.member.5.Namespace` = `aws:elasticbeanstalk:sqsd`
- `OptionSettings.member.5.OptionName` = `HttpConnections`
- `OptionSettings.member.5.Value` = `75`
- `OptionSettings.member.6.Namespace` = `aws:elasticbeanstalk:sqsd`
- `OptionSettings.member.6.OptionName` = `ConnectTimeout`
- `OptionSettings.member.6.Value` = `10`
- `OptionSettings.member.7.Namespace` = `aws:elasticbeanstalk:sqsd`
- `OptionSettings.member.7.OptionName` = `InactivityTimeout`

- `OptionSettings.member.7.Value` = `10`
- `OptionSettings.member.8.Namespace` = `aws:elasticbeanstalk:sqsd`
- `OptionSettings.member.8.OptionName` = `VisibilityTimeout`
- `OptionSettings.member.8.Value` = `60`
- `OptionSettings.member.9.Namespace` = `aws:elasticbeanstalk:sqsd`
- `OptionSettings.member.9.OptionName` = `RetentionPeriod`
- `OptionSettings.member.9.Value` = `345600`

Example

```
https://elasticbeanstalk.us-east-2.amazonaws.com/?ApplicationName=SampleApp
&VersionLabel=Version2
&EnvironmentName=SampleAppEnv2
&TemplateName=MyConfigTemplate
&Description=description
&Tier=Worker
&Operation=CreateEnvironment
&OptionSettings.member.1.Namespace=aws%3Aautoscaling%3Alaunchconfiguration
&OptionSettings.member.1.OptionName=IamInstanceProfile
&OptionSettings.member.1.Value=aws-elasticbeanstalk-ec2-role
&OptionSettings.member.2.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.2.OptionName=WorkerQueueURL
&OptionSettings.member.2.Value=sqsd.elasticbeanstalk.us-east-2.amazonaws.com
&OptionSettings.member.3.Namespace=aws%3elasticbeanstalk%3sqsd
&OptionSettings.member.3.OptionName=HttpPath
&OptionSettings.member.3.Value=%2F
&OptionSettings.member.4.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.4.OptionName=MimeType
&OptionSettings.member.4.Value=application%2Fjson
&OptionSettings.member.5.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.5.OptionName=HttpConnections
&OptionSettings.member.5.Value=75
&OptionSettings.member.6.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.6.OptionName=ConnectTimeout
&OptionSettings.member.6.Value=10
&OptionSettings.member.7.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.7.OptionName=InactivityTimeout
&OptionSettings.member.7.Value=10
&OptionSettings.member.8.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.8.OptionName=VisibilityTimeout
```

```
&OptionSettings.member.8.Value=60
&OptionSettings.member.9.Namespace=aws%3Aelasticbeanstalk%3Asqsd
&OptionSettings.member.9.OptionName=RetentionPeriod
&OptionSettings.member.9.Value=345600
&AuthParams
```

Constructing a Launch Now URL

You can construct a custom URL so that anyone can quickly deploy and run a predetermined web application in AWS Elastic Beanstalk. This URL is called a Launch Now URL. You might need a Launch Now URL, for example, to demonstrate a web application that's built to run on Elastic Beanstalk. With a Launch Now URL, you can use parameters to add the required information to the Create Application wizard in advance. After you add this information to the wizard, anyone can use the URL link to launch an Elastic Beanstalk environment with your web application source in only a few steps. This means users don't need to manually upload or specify the location of the application source bundle. They also don't need to provide any additional information to the wizard.

A Launch Now URL gives Elastic Beanstalk the minimum information that's required to create an application: the application name, solution stack, instance type, and environment type. Elastic Beanstalk uses default values for other configuration details that aren't explicitly specified in your custom Launch Now URL.

A Launch Now URL uses standard URL syntax. For more information, see RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax.

URL parameters

The URL must contain the following parameters, which are case sensitive:

- **region** – Specify an AWS Region. For a list of Regions that are supported by Elastic Beanstalk, see AWS Elastic Beanstalk Endpoints and Quotas in the *AWS General Reference*.
- **applicationName** – Specify the name of your application. Elastic Beanstalk displays the application name in the Elastic Beanstalk console to distinguish it from other applications. By default, the application name also forms the basis of the environment name and environment URL.
- **platform** – Specify the platform version to use for the environment. Use one of the following methods, then URL-encode your choice:

- Specify a platform ARN without a version. Elastic Beanstalk selects the latest platform version of the corresponding platform major version. For example, to select the latest Python 3.6 platform version, specify `Python 3.6 running on 64bit Amazon Linux`.
- Specify the platform name. Elastic Beanstalk selects the latest version of the platform's latest language runtime (for example, `Python`).

For a description of all available platforms and their versions, see Elastic Beanstalk supported platforms.

You can use the AWS Command Line Interface (AWS CLI) to get a list of all the available platform versions with their respective ARNs. The `list-platform-versions` command lists detailed information about all the available platform versions. Use the `--filters` argument to scope down the list. For example, you can scope the list to only show the platform versions of a specific language.

The following example queries all the Python platform versions, and pipes the output through a series of commands. The result is a list of platform version ARNs (without the `/version` tail), in a human-readable format, without URL encoding.

```
$ aws elasticbeanstalk list-platform-versions --filters
 'Type="PlatformName",Operator="contains",Values="Python"' | grep PlatformArn | awk -
F '"' '{print $4}' | awk -F '/' '{print $2}'
Preconfigured Docker - Python 3.4 running on 64bit Debian
Preconfigured Docker - Python 3.4 running on 64bit Debian
Python 2.6 running on 32bit Amazon Linux
Python 2.6 running on 32bit Amazon Linux 2014.03
...
Python 3.6 running on 64bit Amazon Linux
```

The following example adds a Perl command to the last example to URL-encode the output.

```
$ aws elasticbeanstalk list-platform-versions --filters
 'Type="PlatformName",Operator="contains",Values="Python"' | grep PlatformArn | awk
 -F '"' '{print $4}' | awk -F '/' '{print $2}' | perl -MURI::Escape -ne 'chomp;print
 uri_escape($_),"\n"'
Preconfigured%20Docker%20-%20Python%203.4%20running%20on%2064bit%20Debian
Preconfigured%20Docker%20-%20Python%203.4%20running%20on%2064bit%20Debian
Python%202.6%20running%20on%2032bit%20Amazon%20Linux
Python%202.6%20running%20on%2032bit%20Amazon%20Linux%202014.03
...
```

```
Python%203.6%20running%20on%2064bit%20Amazon%20Linux
```

A Launch Now URL can optionally contain the following parameters. If you don't include the optional parameters in your Launch Now URL, Elastic Beanstalk uses default values to create and run your application. When you don't include the **sourceBundleUrl** parameter, Elastic Beanstalk uses the default sample application for the specified **platform**.

- **sourceBundleUrl** – Specify the location of your web application source bundle in URL format. For example, if you uploaded your source bundle to an Amazon S3 bucket, you might specify the value of the **sourceBundleUrl** parameter as `https://amzn-s3-demo-bucket.s3.amazonaws.com/myobject`.

> **Note**
>
> You can specify the value of the **sourceBundleUrl** parameter as an HTTP URL, but the user's web browser will convert characters as needed by applying HTML URL encoding.

- **environmentType** – Specify whether the environment is load balanced and scalable or just a single instance. For more information, see Environment types. You can specify either `LoadBalancing` or `SingleInstance` as the parameter value.
- **tierName** – Specify whether the environment supports a web application that processes web requests or a web application that runs background jobs. For more information, see Elastic Beanstalk worker environments. You can specify either `WebServer` or `Worker`,
- **instanceType** – Specify a server with the characteristics (including memory size and CPU power) that are most appropriate to your application. For more information about Amazon EC2 instance families and types, see Instance types in the *Amazon EC2 User Guide*. For more information about the available instance types across Regions, see Available instance types in the *Amazon EC2 User Guide*.
- **withVpc** – Specify whether to create the environment in an Amazon VPC. You can specify either `true` or `false`. For more information about using Elastic Beanstalk with Amazon VPC, see Using Elastic Beanstalk with Amazon VPC.
- **withRds** – Specify whether to create an Amazon RDS database instance with this environment. For more information, see Using Elastic Beanstalk with Amazon RDS. You can specify either `true` or `false`.

- **rdsDBEngine** – Specify the database engine that you want to use for your Amazon EC2 instances in this environment. You can specify `mysql`, `oracle-sel`, `sqlserver-ex`, `sqlserver-web`, or `sqlserver-se`. The default value is `mysql`.
- **rdsDBAllocatedStorage** – Specify the allocated database storage size in gigabytes (GB). You can specify the following values:
  - **MySQL** – `5` to `1024`. The default is `5`.
  - **Oracle** – `10` to `1024`. The default is `10`.
  - **Microsoft SQL Server Express Edition** – `30`.
  - **Microsoft SQL Server Web Edition** – `30`.
  - **Microsoft SQL Server Standard Edition** – `200`.
- **rdsDBInstanceClass** – Specify the database instance type. The default value is `db.t2.micro` (`db.m1.large` is for an environment that's not running in an Amazon VPC). For a list of database instance classes that are supported by Amazon RDS, see DB Instance Class in the *Amazon Relational Database Service User Guide*.
- **rdsMultiAZDatabase** – Specify whether Elastic Beanstalk needs to create the database instance across multiple Availability Zones. You can specify either `true` or `false`. For more information about multiple Availability Zone deployments with Amazon RDS, see Regions and Availability Zones in the *Amazon Relational Database Service User Guide*.
- **rdsDBDeletionPolicy** – Specify whether to delete or snapshot the database instance on environment termination. You can specify either `Delete` or `Snapshot`.

Example

The following is an example Launch Now URL. After you construct your own, you can give it to your users. For example, you can embed the URL on a webpage or in training materials. When users create an application using the Launch Now URL, the Elastic Beanstalk Create an Application wizard requires no additional input.

```
https://console.aws.amazon.com/elasticbeanstalk/home?region=us-west-2#/newApplication?
applicationName=YourCompanySampleApp
       &platform=PHP%207.3%20running%20on%2064bit%20Amazon%20Linux&sourceBundleUrl=
         http://s3.amazonaws.com/amzn-s3-demo-bucket/
myobject&environmentType=SingleInstance&tierName=WebServer
       &instanceType=m1.small&withVpc=true&withRds=true&rdsDBEngine=

 postgres&rdsDBAllocatedStorage=6&rdsDBInstanceClass=db.m1.small&rdsMultiAZDatabase=
          true&rdsDBDeletionPolicy=Snapshot
```

To use the Launch Now URL

1. Choose the Launch Now URL.
2. After the Elastic Beanstalk console opens, on the **Create a web app** page, choose **Review and launch** to view the settings that Elastic Beanstalk uses to create the application and launch the environment where the application runs.
3. On the **Configure** page, choose **Create app** to create the application.

Creating and updating groups of Elastic Beanstalk environments

With the AWS Elastic Beanstalk ComposeEnvironments API, you can create and update groups of Elastic Beanstalk environments within a single application. Each environment in the group can run a separate component of a service-oriented architecture application. The `Compose Environments` API takes a list of application versions and an optional group name. Elastic Beanstalk creates an environment for each application version, or, if the environments already exist, deploys the application versions to them.

Create links between Elastic Beanstalk environments to designate one environment as a dependency of another. When you create a group of environments with the `Compose Environments` API, Elastic Beanstalk creates dependent environments only after their dependencies are up and running. For more information on environment links, see Creating links between Elastic Beanstalk environments.

The `Compose Environments` API uses an environment manifest to store configuration details that are shared by groups of environments. Each component application must have an `env.yaml` configuration file in its application source bundle that specifies the parameters used to create its environment.

`Compose Environments` requires the `EnvironmentName` and `SolutionStack` to be specified in the environment manifest for each component application.

You can use the `Compose Environments` API with the Elastic Beanstalk command line interface (EB CLI), the AWS CLI, or an SDK. See Managing multiple Elastic Beanstalk environments as a group with the EB CLI for EB CLI instructions.

Using the `Compose Environments` API

For example, you could make an application named `Media Library` that lets users upload and manage images and videos stored in Amazon Simple Storage Service (Amazon S3). The application has a front-end environment, `front`, that runs a web application that lets users upload and download individual files, view their library, and initiate batch processing jobs.

Instead of processing the jobs directly, the front-end application adds jobs to an Amazon SQS queue. The second environment, `worker`, pulls jobs from the queue and processes them. `worker` uses a G2 instance type that has a high-performance GPU, while `front` can run on a more cost-effective generic instance type.

You would organize the project folder, `Media Library`, into separate directories for each component, with each directory containing an environment definition file (`env.yaml`) with the source code for each:

```
~/workspace/media-library
|-- front
|   `-- env.yaml
`-- worker
    `-- env.yaml
```

The following listings show the `env.yaml` file for each component application.

`~/workspace/media-library/front/env.yaml`

```
EnvironmentName: front+
EnvironmentLinks:
  "WORKERQUEUE" : "worker+"
AWSConfigurationTemplateVersion: 1.1.0.0
EnvironmentTier:
  Name: WebServer
  Type: Standard
SolutionStack: 64bit Amazon Linux 2015.09 v2.0.4 running Java 8
OptionSettings:
  aws:autoscaling:launchconfiguration:
    InstanceType: m4.large
```

`~/workspace/media-library/worker/env.yaml`

```
EnvironmentName: worker+
AWSConfigurationTemplateVersion: 1.1.0.0
EnvironmentTier:
  Name: Worker
  Type: SQS/HTTP
SolutionStack: 64bit Amazon Linux 2015.09 v2.0.4 running Java 8
OptionSettings:
  aws:autoscaling:launchconfiguration:
    InstanceType: g2.2xlarge
```

After creating an application version for the front-end (`front-v1`) and worker (`worker-v1`) application components, you call the `Compose Environments` API with the version names. In this example, we use the AWS CLI to call the API.

```
# Create application versions for each component:
~$ aws elasticbeanstalk create-application-version --application-name media-
library --version-label front-v1 --process --source-bundle S3Bucket="amzn-s3-demo-
bucket",S3Key="front-v1.zip"
  {
    "ApplicationVersion": {
        "ApplicationName": "media-library",
        "VersionLabel": "front-v1",
        "Description": "",
        "DateCreated": "2015-11-03T23:01:25.412Z",
        "DateUpdated": "2015-11-03T23:01:25.412Z",
        "SourceBundle": {
            "S3Bucket": "amzn-s3-demo-bucket",
            "S3Key": "front-v1.zip"
        }
    }
  }
~$ aws elasticbeanstalk create-application-version --application-name media-library
 --version-label worker-v1 --process --source-bundle S3Bucket="amzn-s3-demo-
bucket",S3Key="worker-v1.zip"
  {
    "ApplicationVersion": {
        "ApplicationName": "media-library",
        "VersionLabel": "worker-v1",
        "Description": "",
        "DateCreated": "2015-11-03T23:01:48.151Z",
        "DateUpdated": "2015-11-03T23:01:48.151Z",
        "SourceBundle": {
            "S3Bucket": "amzn-s3-demo-bucket",
            "S3Key": "worker-v1.zip"
        }
    }
  }
# Create environments:
~$ aws elasticbeanstalk compose-environments --application-name media-library --group-
name dev --version-labels front-v1 worker-v1
```

The third call creates two environments, `front-dev` and `worker-dev`. The API creates the names of the environments by concatenating the `EnvironmentName` specified in the `env.yaml` file with the `group name` option specified in the `Compose Environments` call, separated by a hyphen. The total length of these two options and the hyphen must not exceed the maximum allowed environment name length of 23 characters.

The application running in the `front-dev` environment can access the name of the Amazon SQS queue attached to the `worker-dev` environment by reading the `WORKERQUEUE` variable. For more information on environment links, see Creating links between Elastic Beanstalk environments.

Managing multiple Elastic Beanstalk environments as a group with the EB CLI

You can use the EB CLI to create groups of AWS Elastic Beanstalk environments, each running a separate component of a service-oriented architecture application. The EB CLI manages such groups by using the ComposeEnvironments API.

Note

Environment groups are different than multiple containers in a Multicontainer Docker environment. With environment groups, each component of your application runs in a separate Elastic Beanstalk environment, with its own dedicated set of Amazon EC2 instances. Each component can scale separately. With Multicontainer Docker, you combine several components of an application into a single environment. All components share the same set of Amazon EC2 instances, with each instance running multiple Docker containers. Choose one of these architectures according to your application's needs.
For details about Multicontainer Docker, see Using the ECS managed Docker platform branch in Elastic Beanstalk.

Organize your application components into the following folder structure:

```
~/project-name
|-- component-a
|   `-- env.yaml
`-- component-b
    `-- env.yaml
```

Each subfolder contains the source code for an independent component of an application that will run in its own environment and an environment definition file named `env.yaml`. For details on the `env.yaml` format, see Environment manifest (`env.yaml`).

To use the `Compose Environments` API, first run **eb init** from the project folder, specifying each component by the name of the folder that contains it with the `--modules` option:

```
~/workspace/project-name$ eb init --modules component-a component-b
```

The EB CLI prompts you to configure each component, and then creates the `.elasticbeanstalk` directory in each component folder. EB CLI doesn't create configuration files in the parent directory.

```
~/project-name
|-- component-a
|   |-- .elasticbeanstalk
|   `-- env.yaml
`-- component-b
    |-- .elasticbeanstalk
    `-- env.yaml
```

Next, run the **eb create** command with a list of environments to create, one for each component:

```
~/workspace/project-name$ eb create --modules component-a component-b --env-group-
suffix group-name
```

This command creates an environment for each component. The names of the environments are created by concatenating the `EnvironmentName` specified in the `env.yaml` file with the group name, separated by a hyphen. The total length of these two options and the hyphen must not exceed the maximum allowed environment name length of 23 characters.

To update the environment, use the **eb deploy** command:

```
~/workspace/project-name$ eb deploy --modules component-a component-b
```

You can update each component individually or you can update them as a group. Specify the components that you want to update with the `--modules` option.

The EB CLI stores the group name that you used with **eb create** in the `branch-defaults` section of the EB CLI configuration file under `/.elasticbeanstalk/config.yml`. To deploy your application to a different group, use the `--env-group-suffix` option when you run **eb deploy**. If the group does not already exist, the EB CLI will create a new group of environments:

```
~/workspace/project-name$ eb deploy --modules component-a component-b --env-group-
suffix group-2-name
```

To terminate environments, run **eb terminate** in the folder for each module. By default, the EB CLI will show an error if you try to terminate an environment that another running environment is dependent on. Terminate the dependent environment first, or use the `--ignore-links` option to override the default behavior:

```
~/workspace/project-name/component-b$ eb terminate --ignore-links
```

Deploying applications to Elastic Beanstalk environments

You can use the AWS Elastic Beanstalk console to upload an updated source bundle and deploy it to your Elastic Beanstalk environment, or redeploy a previously uploaded version.

Each deployment is identified by a deployment ID. Deployment IDs start at 1 and increment by one with each deployment and instance configuration change. If you enable enhanced health reporting, Elastic Beanstalk displays the deployment ID in both the health console and the EB CLI when it reports instance health status. The deployment ID helps you determine the state of your environment when a rolling update fails.

Elastic Beanstalk provides several deployment policies and settings. For details about configuring a policy and additional settings, see the section called “Deployment options”. The following table lists the policies and the kinds of environments that support them.

Supported deployment policies

Deployment policy	Load-balanced environments	Single-instance environments	Legacy Windows Server environme nts†
All at once	✓ Yes	✓ Yes	✓ Yes
Rolling	✓ Yes	☓ No	✓ Yes
Rolling with an additional batch	✓ Yes	☓ No	☓ No
Immutable	✓ Yes	✓ Yes	☓ No
Traffic splitting	✓ Yes (Application Load Balancer)	☓ No	☓ No

† In this table, a *Legacy Windows Server environment* is an environment based on a Windows Server platform configuration that uses an IIS version earlier than IIS 8.5.

Warning

Some policies replace all instances during the deployment or update. This causes all accumulated Amazon EC2 burst balances to be lost. It happens in the following cases:

- Managed platform updates with instance replacement enabled
- Immutable updates
- Deployments with immutable updates or traffic splitting enabled

Choosing a deployment policy

Choosing the right deployment policy for your application is a tradeoff of a few considerations, and depends on your particular needs. The the section called "Deployment options" page has more information about each policy, and a detailed description of the workings of some of them.

The following list provides summary information about the different deployment policies and adds related considerations.

- **All at once –** The quickest deployment method. Suitable if you can accept a short loss of service, and if quick deployments are important to you. With this method, Elastic Beanstalk deploys the new application version to each instance. Then, the web proxy or application server might need to restart. As a result, your application might be unavailable to users (or have low availability) for a short time.
- **Rolling –** Avoids downtime and minimizes reduced availability, at a cost of a longer deployment time. Suitable if you can't accept any period of completely lost service. With this method, your application is deployed to your environment one batch of instances at a time. Most bandwidth is retained throughout the deployment.
- **Rolling with additional batch –** Avoids any reduced availability, at a cost of an even longer deployment time compared to the *Rolling* method. Suitable if you must maintain the same bandwidth throughout the deployment. With this method, Elastic Beanstalk launches an extra batch of instances, then performs a rolling deployment. Launching the extra batch takes time, and ensures that the same bandwidth is retained throughout the deployment.
- **Immutable –** A slower deployment method, that ensures your new application version is always deployed to new instances, instead of updating existing instances. It also has the additional advantage of a quick and safe rollback in case the deployment fails. With this method, Elastic Beanstalk performs an immutable update to deploy your application. In an immutable update, a second Auto Scaling group is launched in your environment and the new version serves traffic alongside the old version until the new instances pass health checks.
- **Traffic splitting –** A canary testing deployment method. Suitable if you want to test the health of your new application version using a portion of incoming traffic, while keeping the rest of the traffic served by the old application version.

The following table compares deployment method properties.

Deployment methods

Methoc	Impact of failed deployment	Deploy time	Zero downti	No DNS change	Rollbac process	Code deployed to
All at once	Downtime	🕒	✗ No	✓ Yes	Manual redeplo	Existing instances

Methoc	Impact of failed deployment	Deploy time	Zero downti	No DNS change	Rollbac process	Code deployed to
Rolling	Single batch out of service; any successful batches before failure running new application version	🕒 🕒	✓ Yes	✓ Yes	Manual redeplo	Existing instances
Rolling with an additio l batch	Minimal if first batch fails; otherwise, similar to **Rolling**	🕒 🕒	✓ Yes	✓ Yes	Manual redeplo	New and existing instances
Immuta	Minimal	🕒 🕒	✓ Yes	✓ Yes	Termina new instanc	New instances
Traffic splittin	Percentage of client traffic routed to new version temporarily impacted	🕒 🕒	✓ Yes	✓ Yes	Reroute traffic and termina new instanc	New instances
Blue/ green	Minimal	🕒 🕒	✓ Yes	✗ No	Swap URL	New instances

† *Varies depending on batch size.*

†† *Varies depending on **evaluation time** option setting.*

Deploying a new application version

You can perform deployments from your environment's dashboard.

To deploy a new application version to an Elastic Beanstalk environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Upload and deploy**.
4. Use the on-screen form to upload the application source bundle.
5. Choose **Deploy**.

Redeploying a previous version

You can also deploy a previously uploaded version of your application to any of its environments from the application versions page.

To deploy an existing application version to an existing environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.
3. In the navigation pane, find your application's name and choose **Application versions**.
4. Select the application version to deploy.
5. Choose **Actions**, and then choose **Deploy**.
6. Select an environment, and then choose **Deploy**.

Other ways to deploy your application

If you deploy often, consider using the Elastic Beanstalk Command Line Interface (EB CLI) to manage your environments. The EB CLI creates a repository alongside your source code. It can also create a source bundle, upload it to Elastic Beanstalk, and deploy it with a single command.

For deployments that depend on resource configuration changes or a new version that can't run alongside the old version, you can launch a new environment with the new version and perform a CNAME swap for a blue/green deployment.

To automate your build, test, and deployment processes, you can implement continuous integration and continuous deployment (CI/CD) with your Elastic Beanstalk environment. For more information, see Implementing CI/CD integration with your Elastic Beanstalk environment.

Deployment policies and settings

AWS Elastic Beanstalk provides several options for how deployments are processed, including deployment policies (*All at once*, *Rolling*, *Rolling with additional batch*, *Immutable*, and *Traffic splitting*) and options that let you configure batch size and health check behavior during deployments. By default, your environment uses all-at-once deployments. If you created the environment with the EB CLI and it's a scalable environment (you didn't specify the `--single` option), it uses rolling deployments.

With *rolling deployments*, Elastic Beanstalk splits the environment's Amazon EC2 instances into batches and deploys the new version of the application to one batch at a time. It leaves the rest of the instances in the environment running the old version of the application. During a rolling deployment, some instances serve requests with the old version of the application, while instances in completed batches serve other requests with the new version. For details, see the section called "How rolling deployments work".

To maintain full capacity during deployments, you can configure your environment to launch a new batch of instances before taking any instances out of service. This option is known as a *rolling deployment with an additional batch*. When the deployment completes, Elastic Beanstalk terminates the additional batch of instances.

Immutable deployments perform an immutable update to launch a full set of new instances running the new version of the application in a separate Auto Scaling group, alongside the instances running the old version. Immutable deployments can prevent issues caused by partially completed rolling deployments. If the new instances don't pass health checks, Elastic Beanstalk terminates them, leaving the original instances untouched.

Traffic-splitting deployments let you perform canary testing as part of your application deployment. In a traffic-splitting deployment, Elastic Beanstalk launches a full set of new instances just like during an immutable deployment. It then forwards a specified percentage of incoming client traffic to the new application version for a specified evaluation period. If the new instances stay healthy, Elastic Beanstalk forwards all traffic to them and terminates the old ones. If the new instances don't pass health checks, or if you choose to abort the deployment, Elastic Beanstalk moves traffic back to the old instances and terminates the new ones. There's never any service interruption. For details, see the section called "How traffic-splitting deployments work".

Warning

Some policies replace all instances during the deployment or update. This causes all accumulated Amazon EC2 burst balances to be lost. It happens in the following cases:

- Managed platform updates with instance replacement enabled
- Immutable updates
- Deployments with immutable updates or traffic splitting enabled

If your application doesn't pass all health checks, but still operates correctly at a lower health status, you can allow instances to pass health checks with a lower status, such as `Warning`, by modifying the **Healthy threshold** option. If your deployments fail because they don't pass health checks and you need to force an update regardless of health status, specify the **Ignore health check** option.

When you specify a batch size for rolling updates, Elastic Beanstalk also uses that value for rolling application restarts. Use rolling restarts when you need to restart the proxy and application servers running on your environment's instances without downtime.

Configuring application deployments

In the environment management console, enable and configure batched application version deployments by editing **Updates and Deployments** on the environment's **Configuration** page.

To configure deployments (console)

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Rolling updates and deployments** configuration category, choose **Edit**.
5. In the **Application Deployments** section, choose a **Deployment policy**, batch settings, and health check options.
6. To save the changes choose **Apply** at the bottom of the page.

The **Application deployments** section of the **Rolling updates and deployments** page has the following options for application deployments:

- **Deployment policy –** Choose from the following deployment options:
  - **All at once –** Deploy the new version to all instances simultaneously. All instances in your environment are out of service for a short time while the deployment occurs.
  - **Rolling –** Deploy the new version in batches. Each batch is taken out of service during the deployment phase, reducing your environment's capacity by the number of instances in a batch.
  - **Rolling with additional batch –** Deploy the new version in batches, but first launch a new batch of instances to ensure full capacity during the deployment process.
  - **Immutable –** Deploy the new version to a fresh group of instances by performing an immutable update.
  - **Traffic splitting –** Deploy the new version to a fresh group of instances and temporarily split incoming client traffic between the existing application version and the new one.

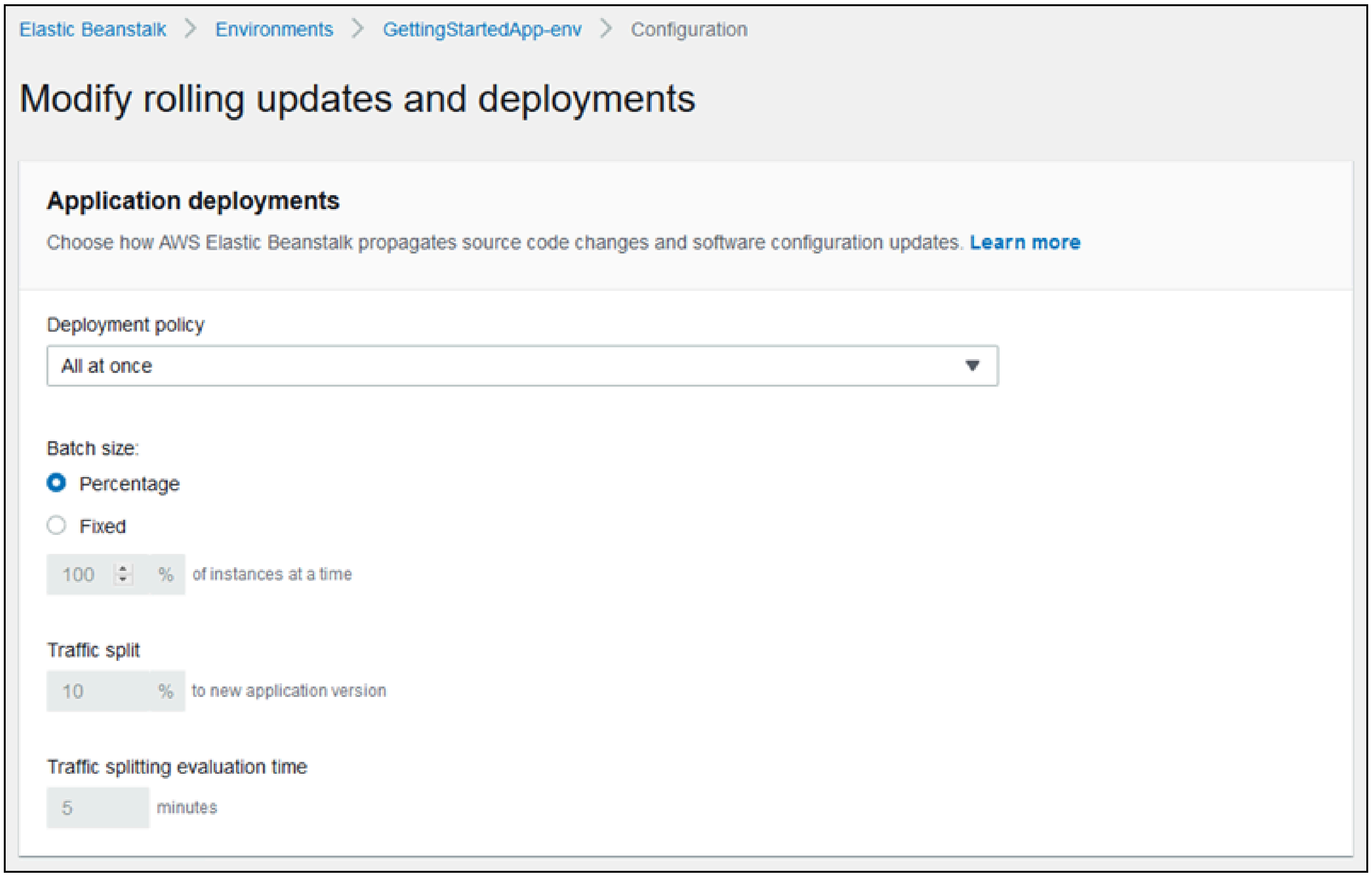
For the **Rolling** and **Rolling with additional batch** deployment policies you can configure:

- **Batch size –** The size of the set of instances to deploy in each batch.

  Choose **Percentage** to configure a percentage of the total number of EC2 instances in the Auto Scaling group (up to 100 percent), or choose **Fixed** to configure a fixed number of instances (up to the maximum instance count in your environment's Auto Scaling configuration).

For the **Traffic splitting** deployment policy you can configure the following:

- **Traffic split –** The initial percentage of incoming client traffic that Elastic Beanstalk shifts to environment instances running the new application version you're deploying.
- **Traffic splitting evaluation time –** The time period, in minutes, that Elastic Beanstalk waits after an initial healthy deployment before proceeding to shift all incoming client traffic to the new application version that you're deploying.



Elastic Beanstalk > Environments > GettingStartedApp-env > Configuration

Modify rolling updates and deployments

Application deployments

Choose how AWS Elastic Beanstalk propagates source code changes and software configuration updates. **Learn more**

Deployment policy: All at once

Batch size: [✓ Percentage] Fixed

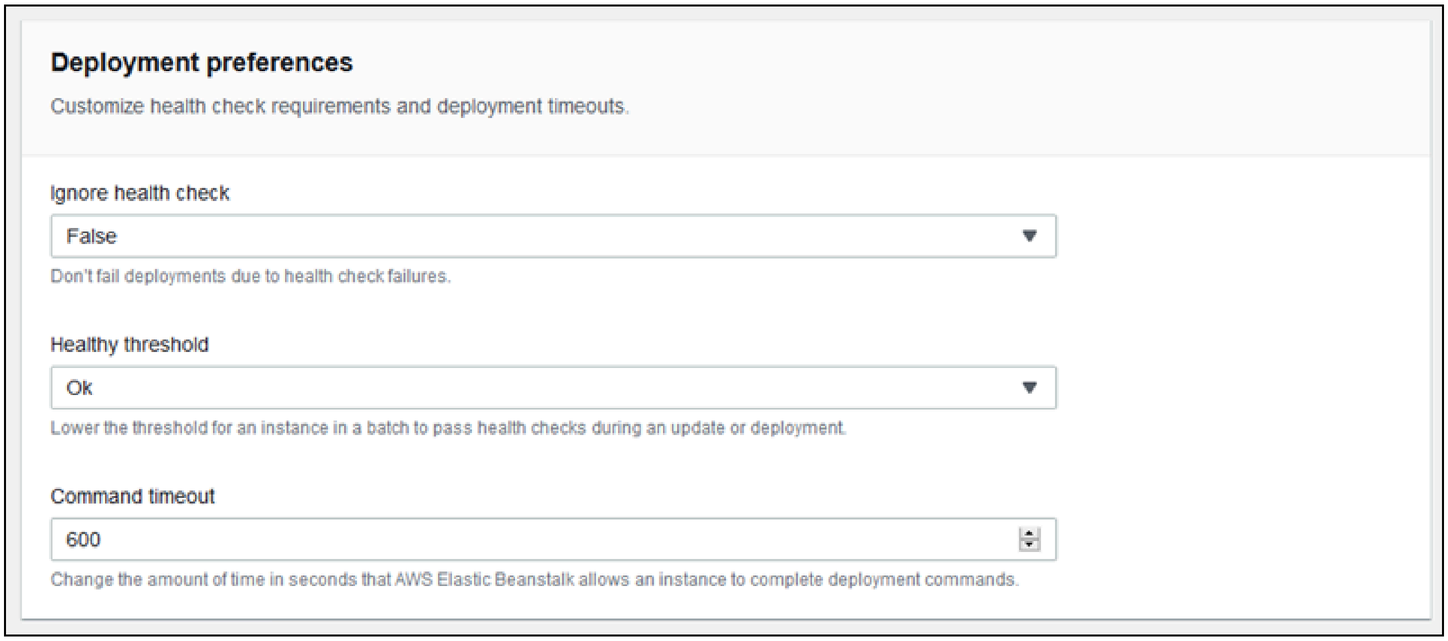
100 % of instances at a time

Traffic split: 10 % to new application version

Traffic splitting evaluation time: 5 minutes

The **Deployment preferences** section contains options related to health checks.

- **Ignore health check** – Prevents a deployment from rolling back when a batch fails to become healthy within the **Command timeout**.
- **Healthy threshold** – Lowers the threshold at which an instance is considered healthy during rolling deployments, rolling updates, and immutable updates.
- **Command timeout** – The number of seconds to wait for an instance to become healthy before canceling the deployment or, if **Ignore health check** is set, to continue to the next batch.



Deployment preferences

Customize health check requirements and deployment timeouts.

Ignore health check

False

Don't fail deployments due to health check failures.

Healthy threshold

Ok

Lower the threshold for an instance in a batch to pass health checks during an update or deployment.

Command timeout

600

Change the amount of time in seconds that AWS Elastic Beanstalk allows an instance to complete deployment commands.

How rolling deployments work

When processing a batch, Elastic Beanstalk detaches all instances in the batch from the load balancer, deploys the new application version, and then reattaches the instances. If you enable connection draining, Elastic Beanstalk drains existing connections from the Amazon EC2 instances in each batch before beginning the deployment.

After reattaching the instances in a batch to the load balancer, Elastic Load Balancing waits until they pass a minimum number of Elastic Load Balancing health checks (the **Healthy check count threshold** value), and then starts routing traffic to them. If no health check URL is configured, this can happen very quickly, because an instance will pass the health check as soon as it can accept a TCP connection. If a health check URL is configured, the load balancer doesn't route traffic to the updated instances until they return a `200 OK` status code in response to an `HTTP GET` request to the health check URL.

Elastic Beanstalk waits until all instances in a batch are healthy before moving on to the next batch. With basic health reporting, instance health depends on the Elastic Load Balancing health check status. When all instances in the batch pass enough health checks to be considered healthy by Elastic Load Balancing, the batch is complete. If enhanced health reporting is enabled, Elastic Beanstalk considers several other factors, including the result of incoming requests. With enhanced health reporting, all instances must pass 12 consecutive health checks with an OK status within two minutes for web server environments, and 18 health checks within three minutes for worker environments.

If a batch of instances does not become healthy within the command timeout, the deployment fails. After a failed deployment, check the health of the instances in your environment for information about the cause of the failure. Then perform another deployment with a fixed or known good version of your application to roll back.

If a deployment fails after one or more batches completed successfully, the completed batches run the new version of your application while any pending batches continue to run the old version. You can identify the version running on the instances in your environment on the health page in the console. This page displays the deployment ID of the most recent deployment that executed on each instance in your environment. If you terminate instances from the failed deployment, Elastic Beanstalk replaces them with instances running the application version from the most recent successful deployment.

How traffic-splitting deployments work

Traffic-splitting deployments allow you to perform canary testing. You direct some incoming client traffic to your new application version to verify the application's health before committing to the new version and directing all traffic to it.

During a traffic-splitting deployment, Elastic Beanstalk creates a new set of instances in a separate temporary Auto Scaling group. Elastic Beanstalk then instructs the load balancer to direct a certain percentage of your environment's incoming traffic to the new instances. Then, for a configured amount of time, Elastic Beanstalk tracks the health of the new set of instances. If all is well, Elastic Beanstalk shifts remaining traffic to the new instances and attaches them to the environment's original Auto Scaling group, replacing the old instances. Then Elastic Beanstalk cleans up—terminates the old instances and removes the temporary Auto Scaling group.

Note

The environment's capacity doesn't change during a traffic-splitting deployment. Elastic Beanstalk launches the same number of instances in the temporary Auto Scaling group as there are in the original Auto Scaling group at the time the deployment starts. It then maintains a constant number of instances in both Auto Scaling groups for the deployment duration. Take this fact into account when configuring the environment's traffic splitting evaluation time.

Rolling back the deployment to the previous application version is quick and doesn't impact service to client traffic. If the new instances don't pass health checks, or if you choose to abort the deployment, Elastic Beanstalk moves traffic back to the old instances and terminates the new ones. You can abort any deployment by using the environment overview page in the Elastic Beanstalk console, and choosing **Abort current operation** in **Environment actions**. You can also call the AbortEnvironmentUpdate API or the equivalent AWS CLI command.

Traffic-splitting deployments require an Application Load Balancer. Elastic Beanstalk uses this load balancer type by default when you create your environment using the Elastic Beanstalk console or the EB CLI.

Deployment option namespaces

You can use the configuration options in the `aws:elasticbeanstalk:command` namespace to configure your deployments. If you choose the traffic-splitting policy, additional options for this policy are available in the `aws:elasticbeanstalk:trafficsplitting` namespace.

Use the `DeploymentPolicy` option to set the deployment type. The following values are supported:

- `AllAtOnce` – Disables rolling deployments and always deploys to all instances simultaneously.
- `Rolling` – Enables standard rolling deployments.
- `RollingWithAdditionalBatch` – Launches an extra batch of instances, before starting the deployment, to maintain full capacity.
- `Immutable` – Performs an immutable update for every deployment.
- `TrafficSplitting` – Performs traffic-splitting deployments to canary-test your application deployments.

When you enable rolling deployments, set the `BatchSize` and `BatchSizeType` options to configure the size of each batch. For example, to deploy 25 percent of all instances in each batch, specify the following options and values.

Example .ebextensions/rolling-updates.config

```
option_settings:
  aws:elasticbeanstalk:command:
    DeploymentPolicy: Rolling
    BatchSizeType: Percentage
    BatchSize: 25
```

To deploy to five instances in each batch, regardless of the number of instances running, and to bring up an extra batch of five instances running the new version before pulling any instances out of service, specify the following options and values.

Example .ebextensions/rolling-additionalbatch.config

```
option_settings:
  aws:elasticbeanstalk:command:
    DeploymentPolicy: RollingWithAdditionalBatch
    BatchSizeType: Fixed
    BatchSize: 5
```

To perform an immutable update for each deployment with a health check threshold of **Warning**, and proceed with the deployment even if instances in a batch don't pass health checks within a timeout of 15 minutes, specify the following options and values.

Example .ebextensions/immutable-ignorehealth.config

```
option_settings:
  aws:elasticbeanstalk:command:
    DeploymentPolicy: Immutable
    HealthCheckSuccessThreshold: Warning
    IgnoreHealthCheck: true
    Timeout: "900"
```

To perform traffic-splitting deployments, forwarding 15 percent of client traffic to the new application version and evaluating health for 10 minutes, specify the following options and values.

Example .ebextensions/traffic-splitting.config

```
option_settings:
  aws:elasticbeanstalk:command:
    DeploymentPolicy: TrafficSplitting
  aws:elasticbeanstalk:trafficsplitting:
    NewVersionPercent: "15"
    EvaluationTime: "10"
```

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See Recommended values for details.

Blue/Green deployments with Elastic Beanstalk

Because AWS Elastic Beanstalk performs an in-place update when you update your application versions, your application might become unavailable to users for a short period of time. To avoid this, perform a blue/green deployment. To do this, deploy the new version to a separate environment, and then swap the CNAMEs of the two environments to redirect traffic to the new version instantly.

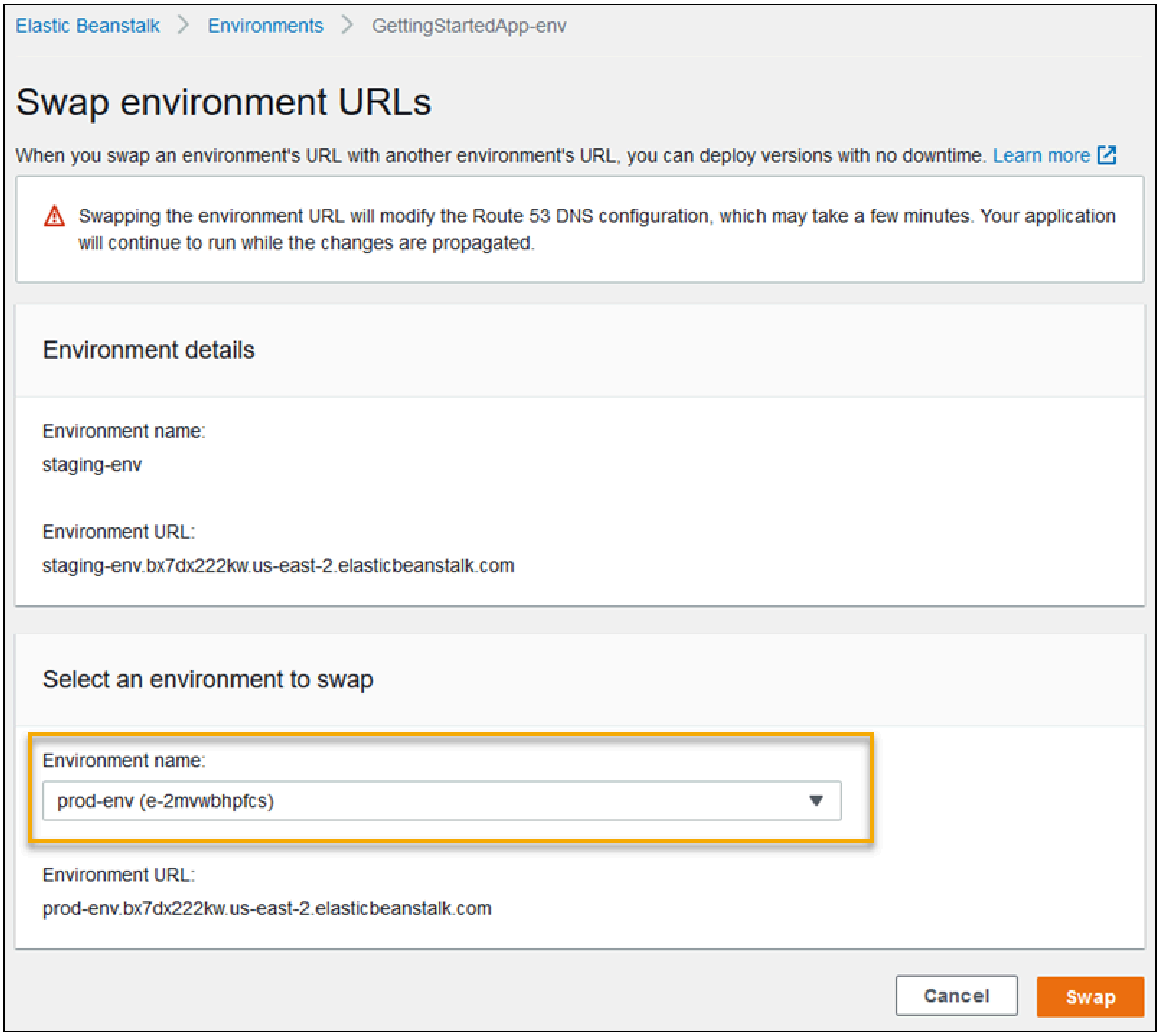
A blue/green deployment is also required if you want to update an environment to an incompatible platform version. For more information, see the section called "Platform updates".

Blue/green deployments require that your environment runs independently of your production database, if your application uses one. If your environment includes a database that Elastic Beanstalk created on your behalf, the database and connection of the environment isn't preserved unless you take specific actions. If you have a database that you want to retain, use one of the Elastic Beanstalk database lifecycle options. You can choose the Retain option to keep the database and environment operational after decoupling the database. For more information see Database lifecycle in the *Configuring environments* chapter of this guide.

For instructions on how to configure your application to connect to an Amazon RDS instance that's not managed by Elastic Beanstalk, see Using Elastic Beanstalk with Amazon RDS.

To perform a blue/green deployment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. Clone your current environment, or launch a new environment to run the platform version you want.
3. Deploy the new application version to the new environment.
4. Test the new version on the new environment.
5. On the environment overview page, choose **Actions**, and then choose **Swap environment URLs**.
6. For **Environment name**, select the current environment.



Elastic Beanstalk > Environments > GettingStartedApp-env

Swap environment URLs

When you swap an environment's URL with another environment's URL, you can deploy versions with no downtime. Learn more

Swapping the environment URL will modify the Route 53 DNS configuration, which may take a few minutes. Your application will continue to run while the changes are propagated.

Environment details

Environment name:
staging-env

Environment URL:
staging-env.bx7dx222kw.us-east-2.elasticbeanstalk.com

Select an environment to swap

Environment name:
prod-env (e-2mvwbhpfcs)

Environment URL:
prod-env.bx7dx222kw.us-east-2.elasticbeanstalk.com

7. Choose **Swap**.

Elastic Beanstalk swaps the CNAME records of the old and new environments, redirecting traffic from the old version to the new version.

After Elastic Beanstalk completes the swap operation, verify that the new environment responds when you try to connect to the old environment URL. However, do not terminate your old environment until the DNS changes are propagated and your old DNS records expire. DNS servers don't always clear old records from their cache based on the time to live (TTL) that you set on your DNS records.

Implementing CI/CD integration with your Elastic Beanstalk environment

Elastic Beanstalk integrates with many CI/CD tools to automate your application development workflow. CI/CD practices enable you to automatically build, test, and deploy your applications with minimal manual intervention. Continuous delivery/deployment (CD) extends continuous integration (CI) by automating the deployment process. You can create streamlined deployment pipelines using AWS services like CodePipeline or third-party tools such as Jenkins and GitLab to ensure consistent, reliable deployments to your Elastic Beanstalk environments.

AWS sources to get started

The following list highlights CI/CD tools and the corresponding AWS resources that provide step-by-step guidance for creating automated deployment pipelines to Elastic Beanstalk environments:

- **AWS CodePipeline –** This AWS Getting Started Resource Center tutorial shows you how to set up a continuous deployment pipeline to Elastic Beanstalk from GitHub , S3, or AWS CodeCommit.
- **GitHub Actions –** This .NET on AWS Blog post walks you through configuring YAML-based workflows to setup a continuous deployment pipeline to Elastic Beanstalk directly from GitHub.
- **GitLab –** This AWS DevOps Developer Productivity Blog post demonstrates how to configure GitLab continuous pipelines to deploy Node.js applications to Elastic Beanstalk Docker environments.
- **Azure DevOps –** This .NET on AWS Blog post guides you through implementing a continuous deployment pipeline from an Azure DevOps Git repository to Elastic Beanstalk using Azure Pipelines.

Additional resources

The following third-party tools and resources can help you implement automated deployment pipelines to Elastic Beanstalk environments:

- **Jenkins –** The AWS EBDeployment Jenkins plugin enables direct deployment to Elastic Beanstalk environments from your Jenkins Job Configuration page.
- **Circle CI: –** The Orbs for Elastic Beanstalk provide reusable configuration packages to deploy and scale applications to Elastic Beanstalk.

- **Bitbucket Pipelines** – The article Deploy Elastic Beanstalk Application using Bitbucket Pipelines provides a basic configuration example for implementing Bitbucket Pipelines with Elastic Beanstalk.

Configuration changes

When you modify configuration option settings in the **Configuration** section of the environment management console, AWS Elastic Beanstalk propagates the change to all affected resources. These resources include the load balancer that distributes traffic to the Amazon EC2 instances running your application, the Auto Scaling group that manages those instances, and the EC2 instances themselves.

Many configuration changes can be applied to a running environment without replacing existing instances. For example, setting a health check URL triggers an environment update to modify the load balancer settings, but doesn't cause any downtime because the instances running your application continue serving requests while the update is propagated.

Configuration changes that modify the launch configuration or VPC settings require terminating all instances in your environment and replacing them. For example, when you change the instance type or SSH key setting for your environment, the EC2 instances must be terminated and replaced. Elastic Beanstalk provides several policies that determine how this replacement is done.

- **Rolling updates** – Elastic Beanstalk applies your configuration changes in batches, keeping a minimum number of instances running and serving traffic at all times. This approach prevents downtime during the update process. For details, see Rolling updates.
- **Immutable updates** – Elastic Beanstalk launches a temporary Auto Scaling group outside of your environment with a separate set of instances running with the new configuration. Then Elastic Beanstalk places these instances behind your environment's load balancer. Old and new instances both serve traffic until the new instances pass health checks. At that time, Elastic Beanstalk moves the new instances into your environment's Auto Scaling group and terminates the temporary group and old instances. For details, see Immutable updates.
- **Disabled** – Elastic Beanstalk makes no attempt to avoid downtime. It terminates your environment's existing instances and replaces them with new instances running with the new configuration.

Warning

Some policies replace all instances during the deployment or update. This causes all accumulated Amazon EC2 burst balances to be lost. It happens in the following cases:

- Managed platform updates with instance replacement enabled
- Immutable updates
- Deployments with immutable updates or traffic splitting enabled

Supported update types

Rolling update setting	Load-balanced environments	Single-instance environments	Legacy Windows server environments†
Disabled	✓ Yes	✓ Yes	✓ Yes
Rolling Based on Health	✓ Yes	✗ No	✓ Yes
Rolling Based on Time	✓ Yes	✗ No	✓ Yes
Immutable	✓ Yes	✓ Yes	✗ No

† For the purpose of this table, a *Legacy Windows Server Environment* is an environment based on a Windows Server platform configuration that use an IIS version earlier than IIS 8.5.

Topics

- Elastic Beanstalk rolling environment configuration updates
- Immutable environment updates

Elastic Beanstalk rolling environment configuration updates

When a configuration change requires replacing instances, Elastic Beanstalk can perform the update in batches to avoid downtime while the change is propagated. During a rolling update, capacity is only reduced by the size of a single batch, which you can configure. Elastic Beanstalk takes one batch of instances out of service, terminates them, and then launches a batch with the new configuration. After the new batch starts serving requests, Elastic Beanstalk moves on to the next batch.

Rolling configuration update batches can be processed periodically (time-based), with a delay between each batch, or based on health. For time-based rolling updates, you can configure the amount of time that Elastic Beanstalk waits after completing the launch of a batch of instances before moving on to the next batch. This pause time allows your application to bootstrap and start serving requests.

With health-based rolling updates, Elastic Beanstalk waits until instances in a batch pass health checks before moving on to the next batch. The health of an instance is determined by the health reporting system, which can be basic or enhanced. With basic health, a batch is considered healthy as soon as all instances in it pass Elastic Load Balancing (ELB) health checks.

With enhanced health reporting, all of the instances in a batch must pass multiple consecutive health checks before Elastic Beanstalk will move on to the next batch. In addition to ELB health checks, which check only your instances, enhanced health monitors application logs and the state of your environment's other resources. In a web server environment with enhanced health, all instances must pass 12 health checks over the course of two minutes (18 checks over three minutes for worker environments). If any instance fails one health check, the count resets.

If a batch doesn't become healthy within the rolling update timeout (default is 30 minutes), the update is canceled. Rolling update timeout is a configuration option that is available in the `aws:autoscaling:updatepolicy:rollingupdate` namespace. If your application doesn't pass health checks with `Ok` status but is stable at a different level, you can set the `HealthCheckSuccessThreshold` option in the `aws:elasticbeanstalk:healthreporting:system` namespace to change the level at which Elastic Beanstalk considers an instance to be healthy.

If the rolling update process fails, Elastic Beanstalk starts another rolling update to roll back to the previous configuration. A rolling update can fail due to failed health checks or if launching new instances causes you to exceed the quotas on your account. If you hit a quota on the number of Amazon EC2 instances, for example, the rolling update can fail when it attempts to provision a batch of new instances. In this case, the rollback fails as well.

A failed rollback ends the update process and leaves your environment in an unhealthy state. Unprocessed batches are still running instances with the old configuration, while any batches that completed successfully have the new configuration. To fix an environment after a failed rollback, first resolve the underlying issue that caused the update to fail, and then initiate another environment update.

An alternative method is to deploy the new version of your application to a different environment and then perform a CNAME swap to redirect traffic with zero downtime. See Blue/Green deployments with Elastic Beanstalk for more information.

Rolling updates versus rolling deployments

Rolling updates occur when you change settings that require new Amazon EC2 instances to be provisioned for your environment. This includes changes to the Auto Scaling group configuration, such as instance type and key-pair settings, and changes to VPC settings. In a rolling update, each batch of instances is terminated before a new batch is provisioned to replace it.

Rolling deployments occur whenever you deploy your application and can typically be performed without replacing instances in your environment. Elastic Beanstalk takes each batch out of service, deploys the new application version, and then places it back in service.

The exception to this is if you change settings that require instance replacement at the same time you deploy a new application version. For example, if you change the key name settings in a configuration file in your source bundle and deploy it to your environment, you trigger a rolling update. Instead of deploying your new application version to each batch of existing instances, a new batch of instances is provisioned with the new configuration. In this case, a separate deployment doesn't occur because the new instances are brought up with the new application version.

Anytime new instances are provisioned as part of an environment update, there is a deployment phase where your application's source code is deployed to the new instances and any configuration settings that modify the operating system or software on the instances are applied. Deployment health check settings (**Ignore health check**, **Healthy threshold**, and **Command timeout**) also apply to health-based rolling updates and immutable updates during the deployment phase.

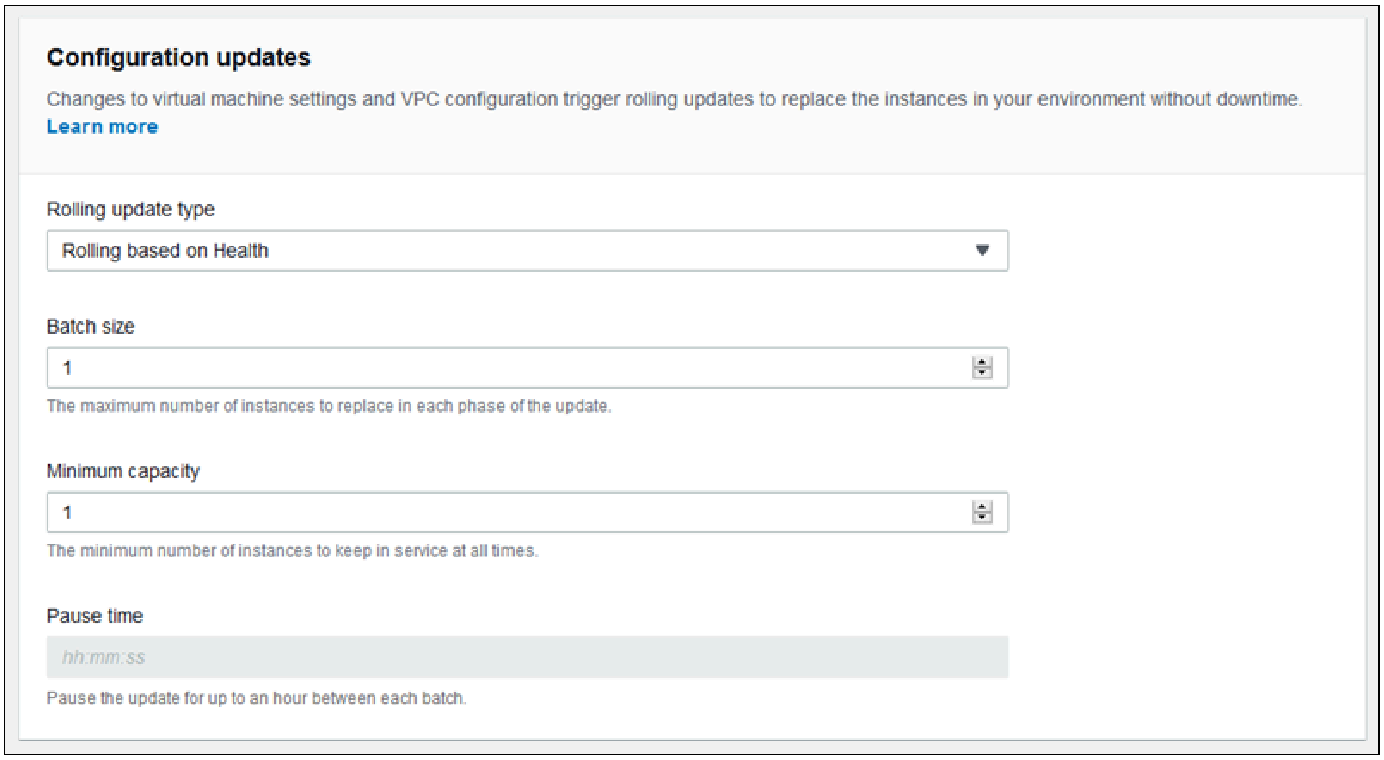
Configuring rolling updates

You can enable and configure rolling updates in the Elastic Beanstalk console.

To enable rolling updates

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Rolling updates and deployments** configuration category, choose **Edit**.
5. In the **Configuration updates** section, for **Rolling update type**, select one of the **Rolling** options.



Configuration updates

Changes to virtual machine settings and VPC configuration trigger rolling updates to replace the instances in your environment without downtime.
Learn more

Rolling update type
Rolling based on Health

Batch size
1
The maximum number of instances to replace in each phase of the update.

Minimum capacity
1
The minimum number of instances to keep in service at all times.

Pause time
hh:mm:ss
Pause the update for up to an hour between each batch.

6. Choose **Batch size**, **Minimum capacity**, and **Pause time** settings.
7. To save the changes choose **Apply** at the bottom of the page.

The **Configuration updates** section of the **Rolling updates and deployments** page has the following options for rolling updates:

- **Rolling update type** – Elastic Beanstalk waits after it finishes updating a batch of instances before moving on to the next batch, to allow those instances to finish bootstrapping and start serving traffic. Choose from the following options:
  - **Rolling based on Health** – Wait until instances in the current batch are healthy before placing instances in service and starting the next batch.
  - **Rolling based on Time** – Specify an amount of time to wait between launching new instances and placing them in service before starting the next batch.

  - **Immutable** – Apply the configuration change to a fresh group of instances by performing an immutable update.
- **Batch size** – The number of instances to replace in each batch, between **1** and **10000**. By default, this value is one-third of the minimum size of the Auto Scaling group, rounded up to a whole number.
- **Minimum capacity** – The minimum number of instances to keep running while other instances are updated, between **0** and **9999**. The default value is either the minimum size of the Auto Scaling group or one less than the maximum size of the Auto Scaling group, whichever number is lower.
- **Pause time** (time-based only) – The amount of time to wait after a batch is updated before moving on to the next batch, to allow your application to start receiving traffic. Between 0 seconds and one hour.

The aws:autoscaling:updatepolicy:rollingupdate namespace

You can also use the configuration options in the `aws:autoscaling:updatepolicy:rollingupdate` namespace to configure rolling updates.

Use the `RollingUpdateEnabled` option to enable rolling updates, and `RollingUpdateType` to choose the update type. The following values are supported for `RollingUpdateType`:

- `Health` – Wait until instances in the current batch are healthy before placing instances in service and starting the next batch.
- `Time` – Specify an amount of time to wait between launching new instances and placing them in service before starting the next batch.
- `Immutable` – Apply the configuration change to a fresh group of instances by performing an immutable update.

When you enable rolling updates, set the `MaxBatchSize` and `MinInstancesInService` options to configure the size of each batch. For time-based and health-based rolling updates, you can also configure a `PauseTime` and `Timeout`, respectively.

For example, to launch up to five instances at a time, while maintaining at least two instances in service, and wait five minutes and 30 seconds between batches, specify the following options and values.

Example .ebextensions/timebased.config

```
option_settings:
  aws:autoscaling:updatepolicy:rollingupdate:
    RollingUpdateEnabled: true
    MaxBatchSize: 5
    MinInstancesInService: 2
    RollingUpdateType: Time
    PauseTime: PT5M30S
```

To enable health-based rolling updates, with a 45-minute timeout for each batch, specify the following options and values.

Example .ebextensions/healthbased.config

```
option_settings:
  aws:autoscaling:updatepolicy:rollingupdate:
    RollingUpdateEnabled: true
    MaxBatchSize: 5
    MinInstancesInService: 2
    RollingUpdateType: Health
    Timeout: PT45M
```

`Timeout` and `PauseTime` values must be specified in ISO8601 duration: PT#H#M#S, where each # is the number of hours, minutes, or seconds, respectively.

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See Recommended values for details.

Immutable environment updates

Immutable environment updates are an alternative to rolling updates. Immutable environment updates ensure that configuration changes that require replacing instances are applied efficiently and safely. If an immutable environment update fails, the rollback process requires only terminating an Auto Scaling group. A failed rolling update, on the other hand, requires performing an additional rolling update to roll back the changes.

To perform an immutable environment update, Elastic Beanstalk creates a second, temporary Auto Scaling group behind your environment's load balancer to contain the new instances. First, Elastic Beanstalk launches a single instance with the new configuration in the new group. This instance serves traffic alongside all of the instances in the original Auto Scaling group that are running the previous configuration.

When the first instance passes health checks, Elastic Beanstalk launches additional instances with the new configuration, matching the number of instances running in the original Auto Scaling group. When all of the new instances pass health checks, Elastic Beanstalk transfers them to the original Auto Scaling group, and terminates the temporary Auto Scaling group and old instances.

Note

During an immutable environment update, the capacity of your environment doubles for a short time when the instances in the new Auto Scaling group start serving requests and before the original Auto Scaling group's instances are terminated. If your environment has many instances, or you have a low on-demand instance quota, ensure that you have enough capacity to perform an immutable environment update. If you are near the quota, consider using rolling updates instead.

Immutable updates require enhanced health reporting to evaluate your environment's health during the update. Enhanced health reporting combines standard load balancer health checks with instance monitoring to ensure that the instances running the new configuration are serving requests successfully.

You can also use immutable updates to deploy new versions of your application, as an alternative to rolling deployments. When you configure Elastic Beanstalk to use immutable updates for application deployments, it replaces all instances in your environment every time you deploy a new version of your application. If an immutable application deployment fails, Elastic Beanstalk reverts the changes immediately by terminating the new Auto Scaling group. This can prevent partial fleet deployments, which can occur when a rolling deployment fails after some batches have already completed.

Warning

Some policies replace all instances during the deployment or update. This causes all accumulated Amazon EC2 burst balances to be lost. It happens in the following cases:

- Managed platform updates with instance replacement enabled
- Immutable updates

- Deployments with immutable updates or traffic splitting enabled

If an immutable update fails, the new instances upload bundle logs to Amazon S3 before Elastic Beanstalk terminates them. Elastic Beanstalk leaves logs from a failed immutable update in Amazon S3 for one hour before deleting them, instead of the standard 15 minutes for bundle and tail logs.

Note

If you use immutable updates for application version deployments, but not for configuration, you might encounter an error if you attempt to deploy an application version that contains configuration changes that would normally trigger a rolling update (for example, configurations that change instance type). To avoid this, make the configuration change in a separate update, or configure immutable updates for both deployments and configuration changes.

You can't perform an immutable update in concert with resource configuration changes. For example, you can't change settings that require instance replacement while also updating other settings, or perform an immutable deployment with configuration files that change configuration settings or additional resources in your source code. If you attempt to change resource settings (for example, load balancer settings) and concurrently perform an immutable update, Elastic Beanstalk returns an error.

If your resource configuration changes aren't dependent on your source code change or on instance configuration, perform them in two updates. If they are dependent, perform a blue/green deployment instead.

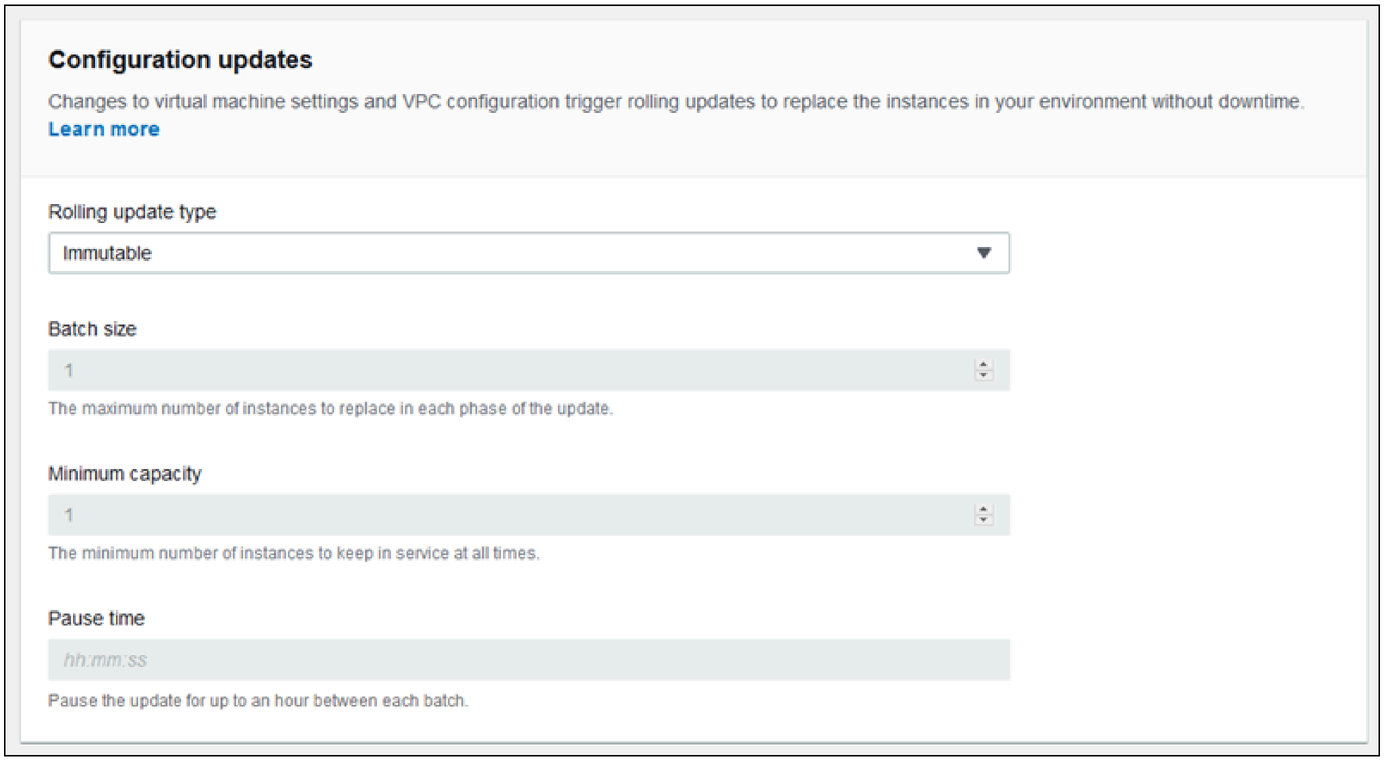
Configuring immutable updates

You can enable and configure immutable updates in the Elastic Beanstalk console.

To enable immutable updates (console)

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

3. In the navigation pane, choose **Configuration**.
4. In the **Rolling updates and deployments** configuration category, choose **Edit**.
5. In the **Configuration Updates** section, set **Rolling update type** to **Immutable**.



Configuration updates

Changes to virtual machine settings and VPC configuration trigger rolling updates to replace the instances in your environment without downtime. Learn more

Rolling update type: Immutable

Batch size: 1
The maximum number of instances to replace in each phase of the update.

Minimum capacity: 1
The minimum number of instances to keep in service at all times.

Pause time: hh:mm:ss
Pause the update for up to an hour between each batch.

6. To save the changes choose **Apply** at the bottom of the page.

The aws:autoscaling:updatepolicy:rollingupdate namespace

You can also use the options in the `aws:autoscaling:updatepolicy:rollingupdate` namespace to configure immutable updates. The following example configuration file enables immutable updates for configuration changes.

Example .ebextensions/immutable-updates.config

```
option_settings:
  aws:autoscaling:updatepolicy:rollingupdate:
    RollingUpdateType: Immutable
```

The following example enables immutable updates for both configuration changes and deployments.

Example .ebextensions/immutable-all.config

```
option_settings:
  aws:autoscaling:updatepolicy:rollingupdate:
    RollingUpdateType: Immutable
  aws:elasticbeanstalk:command:
    DeploymentPolicy: Immutable
```

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See Recommended values for details.

Updating your Elastic Beanstalk environment's platform version

Elastic Beanstalk regularly releases new platform versions to update all Linux-based and Windows Server-based platforms. New platform versions provide updates to existing software components and support for new features and configuration options. To learn about platforms and platform versions, see Elastic Beanstalk platforms glossary.

You can use the Elastic Beanstalk console or the EB CLI to update your environment's platform version. Depending on the platform version you'd like to update to, Elastic Beanstalk recommends one of two methods for performing platform updates.

- Method 1 – Update your environment's platform version. We recommend this method when you're updating to the latest platform version within a platform branch—with the same runtime, web server, application server, and operating system, and without a change in the major platform version. This is the most common and routine platform update.
- Method 2 – Perform a Blue/Green deployment. We recommend this method when you're updating to a platform version in a different platform branch—with a different runtime, web server, application server, or operating system, or to a different major platform version. This is a good approach when you want to take advantage of new runtime capabilities or the latest Elastic Beanstalk functionality, or when you want to move off of a deprecated or retired platform branch.

  Migrating from a legacy platform version requires a blue/green deployment, because these platform versions are incompatible with currently supported versions.

Migrating a Linux application to Amazon Linux 2 requires a blue/green deployment, because Amazon Linux 2 platform versions are incompatible with previous Amazon Linux AMI platform versions.

For more help with choosing the best platform update method, expand the section for your environment's platform.

Docker

Use Method 1 to perform platform updates.

Multicontainer Docker

Use Method 1 to perform platform updates.

Preconfigured Docker

Consider the following cases:

- If you're migrating your application to another platform, for example from *Go 1.4 (Docker)* to *Go 1.11* or from *Python 3.4 (Docker)* to *Python 3.6*, use Method 2.
- If you're migrating your application to a different Docker container version, for example from *Glassfish 4.1 (Docker)* to *Glassfish 5.0 (Docker)*, use Method 2.
- If you're updating to a latest platform version with no change in container version or major version, use Method 1.

Go

Use Method 1 to perform platform updates.

Java SE

Consider the following cases:

- If you're migrating your application to a different Java runtime version, for example from *Java 7* to *Java 8*, use Method 2.
- If you're updating to a latest platform version with no change in runtime version, use Method 1.

Java with Tomcat

Consider the following cases:

- If you're migrating your application to a different Java runtime version or Tomcat application server version, for example from *Java 7 with Tomcat 7* to *Java 8 with Tomcat 8.5*, use Method 2.
- If you're migrating your application across major Java with Tomcat platform versions (v1.x.x, v2.x.x, and v3.x.x), use Method 2.
- If you're updating to a latest platform version with no change in runtime version, application server version, or major version, use Method 1.

.NET on Windows server with IIS

Consider the following cases:

- If you're migrating your application to a different Windows operating system version, for example from *Windows Server 2008 R2* to *Windows Server 2016*, use Method 2.
- If you're migrating your application across major Windows Server platform versions, see Migrating from earlier major versions of the Windows server platform, and use Method 2.
- If your application is currently running on a Windows Server platform V2.x.x and you're updating to a latest platform version, use Method 1.

Note

Windows Server platform versions earlier than v2 aren't semantically versioned. You can only launch the latest version of each of these Windows Server major platform versions and can't roll back after an upgrade.

Node.js

Use Method 2 to perform platform updates.

PHP

Consider the following cases:

- If you're migrating your application to a different PHP runtime version, for example from *PHP 5.6* to *PHP 7.2*, use Method 2.
- If you're migrating your application across major PHP platform versions (v1.x.x and v2.x.x), use Method 2.
- If you're updating to a latest platform version with no change in runtime version or major version, use Method 1.

Python

Consider the following cases:

- If you're migrating your application to a different Python runtime version, for example from *Python 2.7* to *Python 3.6*, use Method 2.
- If you're migrating your application across major Python platform versions (v1.x.x and v2.x.x), use Method 2.
- If you're updating to a latest platform version with no change in runtime version or major version, use Method 1.

Ruby

Consider the following cases:

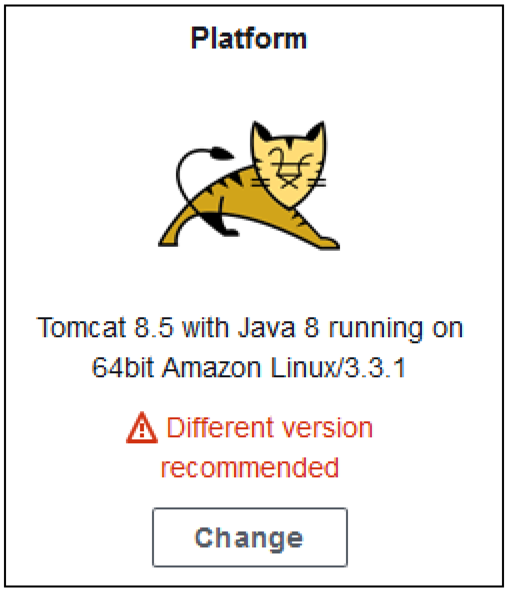
- If you're migrating your application to a different Ruby runtime version or application server version, for example from *Ruby 2.3 with Puma* to *Ruby 2.6 with Puma*, use Method 2.
- If you're migrating your application across major Ruby platform versions (v1.x.x and v2.x.x), use Method 2.
- If you're updating to a latest platform version with no change in runtime version, application server version, or major version, use Method 1.

Method 1 – Update your environment's platform version

Use this method to update to the latest version of your environment's platform branch. If you've previously created an environment using an older platform version, or upgraded your environment from an older version, you can also use this method to revert to a previous platform version, provided that it's in the same platform branch.

To update your environment's platform version

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the environment overview page, under **Platform**, choose **Change**.

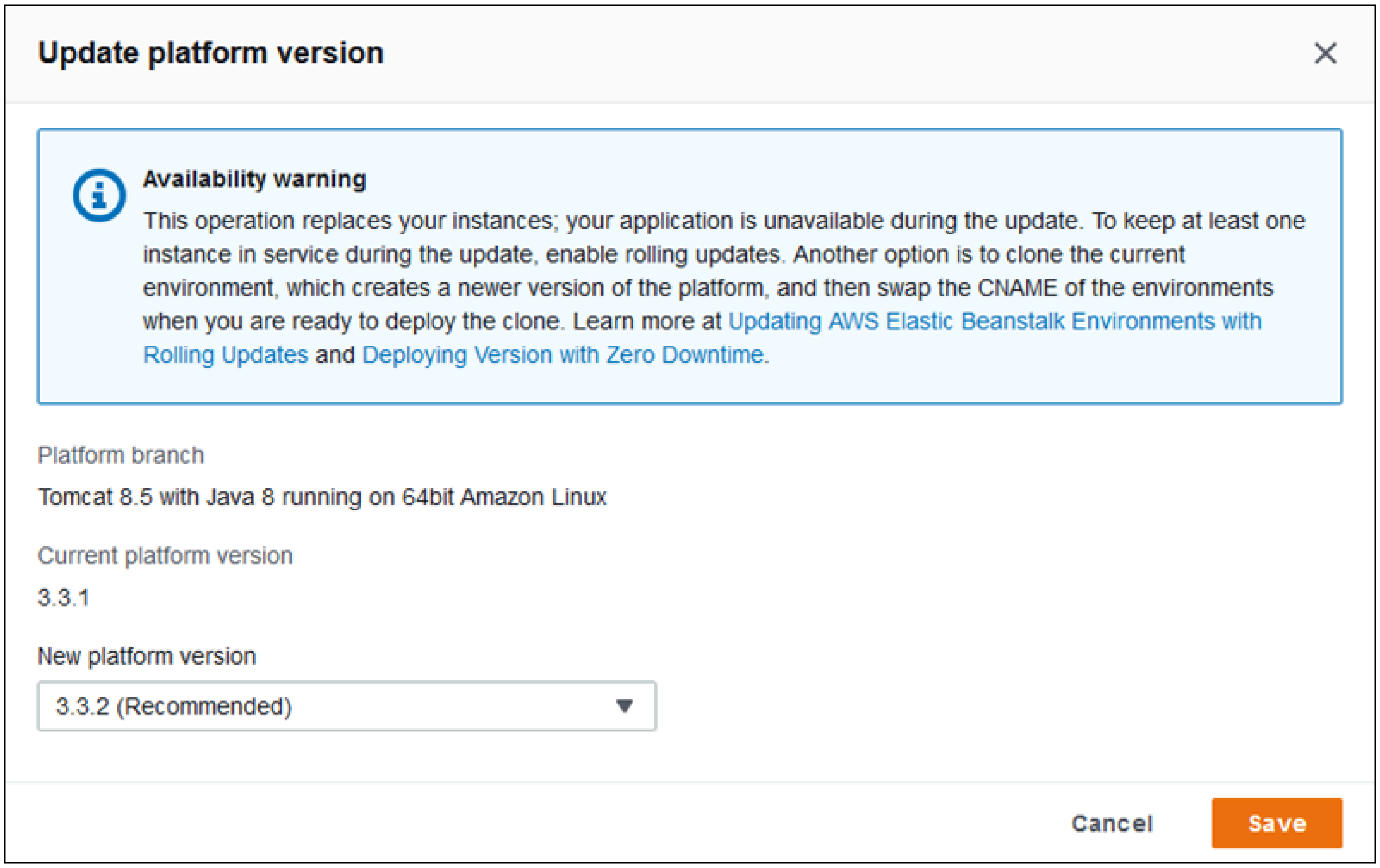


Platform

Tomcat 8.5 with Java 8 running on 64bit Amazon Linux/3.3.1

Different version recommended

4. On the **Update platform version** dialog, select a platform version. The newest (recommended) platform version in the branch is selected automatically. You can update to any version that you've used in the past.



Availability warning

This operation replaces your instances; your application is unavailable during the update. To keep at least one instance in service during the update, enable rolling updates. Another option is to clone the current environment, which creates a newer version of the platform, and then swap the CNAME of the environments when you are ready to deploy the clone. Learn more at Updating AWS Elastic Beanstalk Environments with Rolling Updates and Deploying Version with Zero Downtime.

Platform branch

Tomcat 8.5 with Java 8 running on 64bit Amazon Linux

Current platform version

3.3.1

New platform version

3.3.2 (Recommended)

5. Choose **Save**.

To further simplify platform updates, Elastic Beanstalk can manage them for you. You can configure your environment to apply minor and patch version updates automatically during a configurable weekly maintenance window. Elastic Beanstalk applies managed updates with no downtime or reduction in capacity, and cancels the update immediately if instances running your application on the new version fail health checks. For details, see Managed platform updates.

Method 2 – Perform a Blue/Green deployment

Use this method to update to a different platform branch—with a different runtime, web server, application server, or operating system, or to a different major platform version. This is typically necessary when you want to take advantage of new runtime capabilities or the latest Elastic Beanstalk functionality. It's also required when you're migrating off of a deprecated or retired platform branch.

When you migrate across major platform versions or to platform versions with major component updates, there's a greater likelihood that your application, or some aspects of it, might not function as expected on the new platform version, and might require changes.

Before performing the migration, update your local development machine to the newer runtime versions and other components of the platform you plan on migrating to. Verify that your application still works as expected, and make any necessary code fixes and changes. Then use the following best practice procedure to safely migrate your environment to the new platform version.

To migrate your environment to a platform version with major updates

1. Create a new environment, using the new target platform version, and deploy your application code to it. The new environment should be in the Elastic Beanstalk application that contains the environment you're migrating. Don't terminate the existing environment yet.
2. Use the new environment to migrate your application. In particular:

   - Find and fix any application compatibility issues that you couldn't discover during the development phase.
   - Ensure that any customizations that your application makes using configuration files work correctly in the new environment. These might include option settings, additional installed packages, custom security policies, and script or configuration files installed on environment instances.
   - If your application uses a custom Amazon Machine Image (AMI), create a new custom AMI based on the AMI of the new platform version. To learn more, see Using a custom Amazon machine image (AMI) in your Elastic Beanstalk environment. Specifically, this is required if your application uses the Windows Server platform with a custom AMI, and you're migrating to a Windows Server V2 platform version. In this case, see also Migrating from earlier major versions of the Windows server platform.

   Iterate on testing and deploying your fixes until you're satisfied with the application on the new environment.
3. Turn the new environment into your production environment by swapping its CNAME with the existing production environment's CNAME. For details, see Blue/Green deployments with Elastic Beanstalk.
4. When you're satisfied with the state of your new environment in production, terminate the old environment. For details, see Terminate an Elastic Beanstalk environment.

Managed platform updates

AWS Elastic Beanstalk regularly releases platform updates to provide fixes, software updates, and new features. With managed platform updates, you can configure your environment to automatically upgrade to the latest version of a platform during a scheduled maintenance window. Your application remains in service during the update process with no reduction in capacity. Managed updates are available on both single-instance and load-balanced environments.

Note

This feature isn't available on Windows Server platform versions earlier than version 2 (v2).

You can configure your environment to automatically apply patch version updates, or both patch and minor version updates. Managed platform updates don't support updates across platform branches (updates to different major versions of platform components such as operating system, runtime, or Elastic Beanstalk components), because these can introduce changes that are backward incompatible.

You can also configure Elastic Beanstalk to replace all instances in your environment during the maintenance window, even if a platform update isn't available. Replacing all instances in your environment is helpful if your application encounters bugs or memory issues when running for a long period.

On environments created on November 25, 2019 or later using the Elastic Beanstalk console, managed updates are enabled by default whenever possible. Managed updates require enhanced health to be enabled. Enhanced health is enabled by default when you select one of the configuration presets, and disabled when you select **Custom configuration**. The console can't enable managed updates for older platform versions that don't support enhanced health, or when enhanced health is disabled. When the console enables managed updates for a new environment, the **Weekly update window** is set to a random day of the week at a random time. **Update level** is set to **Minor and patch**, and **Instance replacement** is disabled. You can disable or reconfigure managed updates before the final environment creation step.

For an existing environment, use the Elastic Beanstalk console anytime to configure managed platform updates.

Important

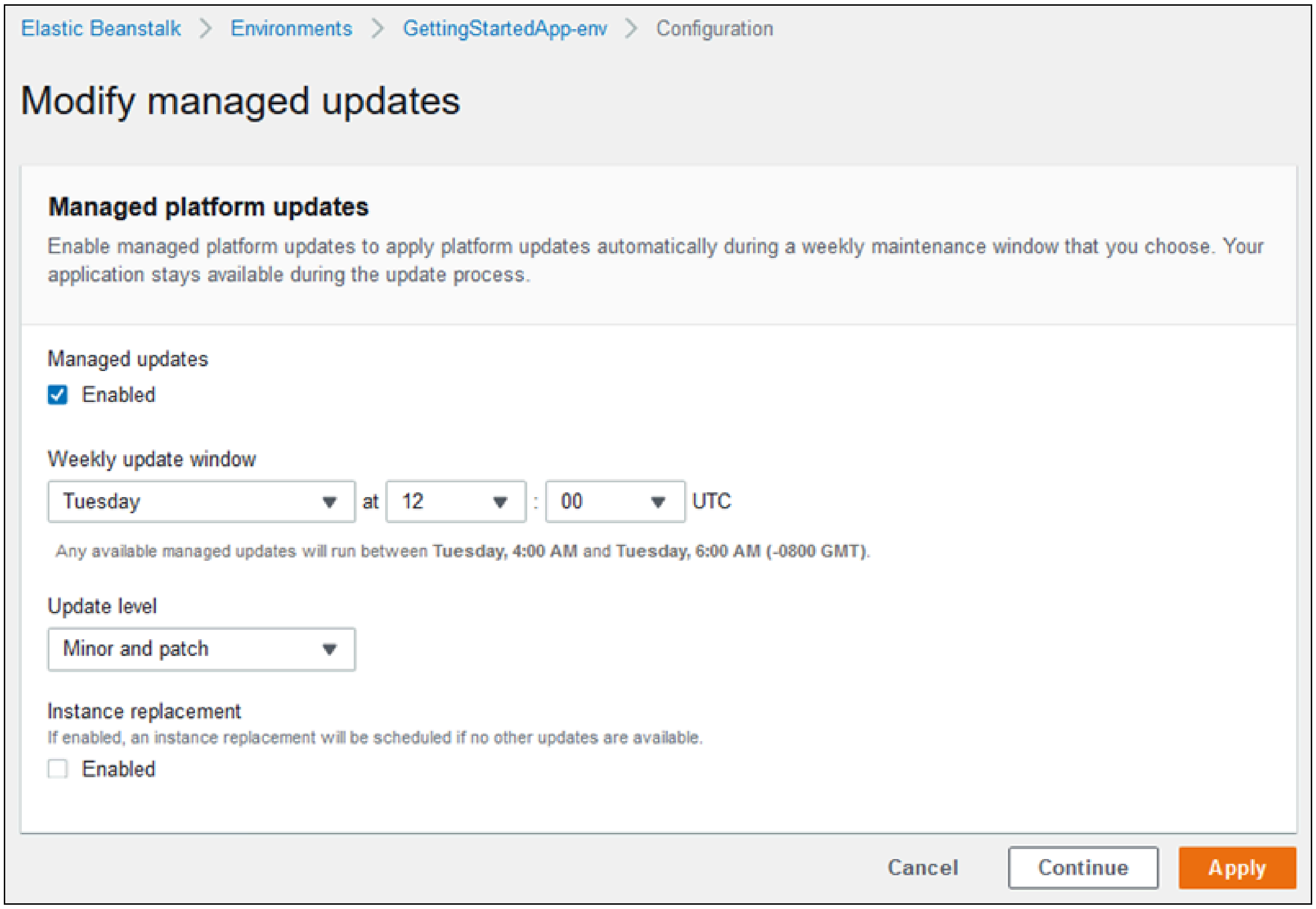
A *high number* of Beanstalk environments in one AWS account may present a risk of throttling issues during managed updates. *High number* is a relative amount that depends on how closely you schedule the managed updates for your environments. Over 200 environments in one account scheduled closely could cause throttling issues, although a lower number may also be problematic.

To balance the resource load for managed updates, we advise that you spread out the scheduled maintenance windows for the environments in one account.

Also, consider a multi-account strategy. For more information, see Organizing Your AWS Environment Using Multiple Accounts on the *AWS Whitepapers & Guides* website.

To configure managed platform updates

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Managed updates** category, choose **Edit**.
5. Disable or enable **Managed updates**.
6. If managed updates are enabled, select a maintenance window, and then select an **Update level**.
7. (Optional) Select **Instance replacement** to enable weekly instance replacement.



Elastic Beanstalk > Environments > GettingStartedApp-env > Configuration

Modify managed updates

Managed platform updates

Enable managed platform updates to apply platform updates automatically during a weekly maintenance window that you choose. Your application stays available during the update process.

Managed updates

☑ Enabled

Weekly update window

Tuesday at 12 : 00 UTC

Any available managed updates will run between **Tuesday, 4:00 AM** and **Tuesday, 6:00 AM (-0800 GMT)**.

Update level

Minor and patch

Instance replacement

If enabled, an instance replacement will be scheduled if no other updates are available.

☐ Enabled

8. To save the changes choose **Apply** at the bottom of the page.

Managed platform updates depend on enhanced health reporting to determine that your application is healthy enough to consider the platform update successful. See Enabling Elastic Beanstalk enhanced health reporting for instructions.

Sections

Permissions required to perform managed platform updates

Elastic Beanstalk needs permission to initiate a platform update on your behalf. To gain these permissions, Elastic Beanstalk assumes the *managed-updates service role*. When you use the default service role for your environment, the Elastic Beanstalk console uses it as the managed-updates service role too. The console assigns the `AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy` managed policy to your service role. This policy has all permissions that Elastic Beanstalk needs to perform managed platform updates.

For details about other ways to set the managed-updates service role, see the section called “Service roles”.

Note

If you use configuration files to extend your environment to include additional resources, you might need to add permissions to your environment's managed-updates service role. Typically you need to add permissions when you reference these resources by name in other sections or files.

If an update fails, you can find the reason for the failure on the Managed updates page.

Managed update maintenance window

When AWS releases a new version of your environment's platform, Elastic Beanstalk schedules a managed platform update during the next weekly maintenance window. Maintenance windows are two hours long. Elastic Beanstalk starts a scheduled update during the maintenance window. The update might not complete until after the window ends.

Note

In most cases, Elastic Beanstalk schedules your managed update to occur during your coming weekly maintenance window. The system considers various aspects of update safety and service availability when scheduling managed updates. In rare cases, an update might not be scheduled for the first coming maintenance window. If this happens, the system tries again during the next maintenance window. To manually apply the managed update, choose **Apply now** as explained in Managing managed updates on this page.

Minor and patch version updates

You can enable managed platform updates to apply patch version updates only, or for both minor and patch version updates. Patch version updates provide bug fixes and performance improvements, and can include minor configuration changes to the on-instance software, scripts, and configuration options. Minor version updates provide support for new Elastic Beanstalk features. You can't apply major version updates, which might make changes that are backward incompatible, with managed platform updates.

In a platform version number, the second number is the minor update version, and the third number is the patch version. For example, a version 2.0.7 platform version has a minor version of 0 and a patch version of 7.

Immutable environment updates

Managed platform updates perform immutable environment updates to upgrade your environment to a new platform version. Immutable updates update your environment without taking any instances out of service or modifying your environment, before confirming that instances running the new version pass health checks.

In an immutable update, Elastic Beanstalk deploys as many instances as are currently running with the new platform version. The new instances begin to take requests alongside those running the old version. If the new set of instances passes all health checks, Elastic Beanstalk terminates the old set of instances, leaving only instances with the new version.

Managed platform updates always perform immutable updates, even when you apply them outside of the maintenance window. If you change the platform version from the **Dashboard**, Elastic Beanstalk applies the update policy that you've chosen for configuration updates.

Warning

Some policies replace all instances during the deployment or update. This causes all accumulated Amazon EC2 burst balances to be lost. It happens in the following cases:

- Managed platform updates with instance replacement enabled
- Immutable updates
- Deployments with immutable updates or traffic splitting enabled

Managing managed updates

The Elastic Beanstalk console shows detailed information about managed updates on the **Managed updates overview** page.

To view information about managed updates (console)

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Managed updates**.

The **Managed updates overview** section provides information about scheduled and pending managed updates. The **History** section lists successful updates and failed attempts.

You can choose to apply a scheduled update immediately, instead of waiting until the maintenance window.

To apply a managed platform update immediately (console)

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Managed updates**.
4. Choose **Apply now**.
5. Verify the update details, and then choose **Apply**.

When you apply a managed platform update outside of the maintenance window, Elastic Beanstalk performs an immutable update. If you update the environment's platform from the Dashboard, or by using a different client, Elastic Beanstalk uses the update type that you selected for configuration changes.

If you don't have a managed update scheduled, your environment might already be running the latest version. Other reasons for not having an update scheduled include:

- A minor version update is available, but your environment is configured to automatically apply only patch version updates.

- Your environment hasn't been scanned since the update was released. Elastic Beanstalk typically checks for updates every hour.
- An update is pending or already in progress.

When your maintenance window starts or when you choose **Apply now**, scheduled updates go into pending status before execution.

Managed action option namespaces

You can use configuration options in the `aws:elasticbeanstalk:managedactions` and `aws:elasticbeanstalk:managedactions:platformupdate` namespaces to enable and configure managed platform updates.

The `ManagedActionsEnabled` option turns on managed platform updates. Set this option to `true` to enable managed platform updates, and use the other options to configure update behavior.

Use `PreferredStartTime` to configure the beginning of the weekly maintenance window in *`day`*:*`hour`*:*`minute`* format.

Set `UpdateLevel` to `minor` or `patch` to apply both minor and patch version updates, or just patch version updates, respectively.

When managed platform updates are enabled, you can enable instance replacement by setting the `InstanceRefreshEnabled` option to `true`. When this setting is enabled, Elastic Beanstalk runs an immutable update on your environment every week, regardless of whether there is a new platform version available.

The following example configuration file enables managed platform updates for patch version updates with a maintenance window starting at 9:00 AM UTC each Tuesday.

Example .ebextensions/managed-platform-update.config

```
option_settings:
  aws:elasticbeanstalk:managedactions:
    ManagedActionsEnabled: true
    PreferredStartTime: "Tue:09:00"
  aws:elasticbeanstalk:managedactions:platformupdate:
    UpdateLevel: patch
    InstanceRefreshEnabled: true
```

Migrating your application from a legacy platform version

If you have deployed an Elastic Beanstalk application that uses a legacy platform version, you should migrate your application to a new environment using a non-legacy platform version so that you can get access to new features. If you are unsure whether you are running your application using a legacy platform version, you can check in the Elastic Beanstalk console. For instructions, see To check if you are using a legacy platform version.

What new features are legacy platform versions missing?

Legacy platforms do not support the following features:

- Configuration files, as described in the Advanced environment customization with configuration files (`.ebextensions`) topic
- ELB health checks, as described in the Basic health reporting topic
- Instance Profiles, as described in the Managing Elastic Beanstalk instance profiles topic
- VPCs, as described in the Using Elastic Beanstalk with Amazon VPC topic
- Data Tiers, as described in the Adding a database to your Elastic Beanstalk environment topic
- Worker Tiers, as described in the Elastic Beanstalk worker environments topic
- Single Instance Environments, as described in the Environment types topic
- Tags, as described in the Tagging resources in your Elastic Beanstalk environments topic
- Rolling Updates, as described in the Elastic Beanstalk rolling environment configuration updates topic

Why are some platform versions marked legacy?

Some older platform versions do not support the latest Elastic Beanstalk features. These versions are marked **(legacy)** on the environment overview page in the Elastic Beanstalk console.

To check if you are using a legacy platform version

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the environment overview page, view the **Platform** name.

Your application is using a legacy platform version if you see **(legacy)** next to the platform's name.

To migrate your application

1. Deploy your application to a new environment. For instructions, go to Creating an Elastic Beanstalk environment.
2. If you have an Amazon RDS DB Instance, update your database security group to allow access to your EC2 security group for your new environment. For instructions on how to find the name of your EC2 security group using the AWS Management Console, see EC2 security groups. For more information about configuring your EC2 security group, go to the "Authorizing Network Access to an Amazon EC2 Security Group" section of Working with DB Security Groups in the *Amazon Relational Database Service User Guide*.
3. Swap your environment URL. For instructions, go to Blue/Green deployments with Elastic Beanstalk.
4. Terminate your old environment. For instructions, go to Terminate an Elastic Beanstalk environment.

Note

If you use AWS Identity and Access Management (IAM) then you will need to update your policies to include AWS CloudFormation and Amazon RDS (if applicable). For more information, see Using Elastic Beanstalk with AWS Identity and Access Management.

Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2

This section describes how to migrate your application using one of the following migration paths.

- Migrate from an *Amazon Linux 2* platform branch to an *Amazon Linux 2023* platform branch.
- Migrate from an *Amazon Linux AMI (AL1)* platform branch to either an *Amazon Linux 2023* (recommended) or an *Amazon Linux 2* platform branch.

Topics

- Migration from Amazon Linux 2 to Amazon Linux 2023
- Migration from Amazon Linux AMI (AL1) to AL2 or AL2023

Migration from Amazon Linux 2 to Amazon Linux 2023

This topic provides guidance to migrate your application from an Amazon Linux 2 platform branch to an Amazon Linux 2023 platform branch.

Differences and compatibility

Between the Elastic Beanstalk AL2 and AL2023 platforms

There is a high degree of compatibility between Elastic Beanstalk Amazon Linux 2 and Amazon Linux 2023 platforms. Although there are some differences to note:

- **Instance Metadata Service Version 1 (IMDSv1)** – The DisableIMDSv1 option setting defaults to `true` on AL2023 platforms. The default is `false` on AL2 platforms.
- **pkg-repo instance tool** – The pkg-repo tool is not available for environments running on AL2023 platforms. However,you can manually apply package and operating system updates to an AL2023 instance. For more information, see Managing packages and operating system updates in the *Amazon Linux 2023 User Guide*.
- **Apache HTTPd configuration** – The Apache `httpd.conf` file for AL2023 platforms has some configuration settings that are different from those for AL2:
  - Deny access to the server’s entire file system by default. These settings are described in *Protect Server Files by Default* on the Apache website Security Tips page.
  - Stop users from overriding security features you've configured. The configuration denies access to set up of `.htaccess` in all directories, except for those specifically enabled. This setting is described in *Protecting System Settings* on the Apache website Security Tips page. The Apache HTTP Server Tutorial: .htaccess files page states this setting may help improve performance.
  - Deny access to files with name pattern `.ht*`. This setting prevents web clients from viewing `.htaccess` and `.htpasswd` files.

You can change any of the above configuration settings for your environment. For more information, see Configuring Apache HTTPD.

Between the Amazon Linux operating systems

For more information about the differences between the Amazon Linux 2 and Amazon Linux 2023 operating systems, see Comparing Amazon Linux 2 and Amazon Linux 2023 in the *Amazon Linux 2023 User Guide*.

For more information about Amazon Linux 2023, see What is Amazon Linux 2023? in the *Amazon Linux 2023 User Guide*.

General migration process

When you're ready to go to production, Elastic Beanstalk requires a blue/green deployment to perform the upgrade. The following are the general best practice steps that we recommend for migration with a blue/green deployment procedure.

Preparing to test for your migration

Before you deploy your application and start testing, review the information in the prior section Differences and compatibility. Also review the reference cited in that section, Comparing Amazon Linux 2 and Amazon Linux 2023 in the *Amazon Linux 2023 User Guide*. Make a note of the specific information from this content that applies or may apply to your application and configuration set up.

High level migration steps

1. Create a new environment that's based on an AL2023 platform branch.
2. Deploy your application to the target AL2023 environment.

   Your existing production environment will remain active and unaffected, while you iterate through testing and making adjustments to the new environment.
3. Test your application thoroughly in the new environment.
4. When your destination AL2023 environment is ready to go to production, swap the CNAMEs of the two environments to redirect traffic to the new AL2023 environment.

More detailed migration steps and best practices

For a more detailed blue/green deployment procedure, see Blue/Green deployments with Elastic Beanstalk.

For more specific guidance and detailed best practice steps, see Blue/Green method.

More references to help plan your migration

The following references can offer additional information to plan your migration.

- Elastic Beanstalk supported platforms in *AWS Elastic Beanstalk Platforms*
- Retired platform branch history
- the section called “Linux platforms”
- Platform retirement FAQ

Migration from Amazon Linux AMI (AL1) to AL2 or AL2023

If your Elastic Beanstalk application is based on an Amazon Linux AMI platform branch, use this section to learn how to migrate your application's environments to Amazon Linux 2 or Amazon Linux 2023. Previous generation platform branches based on Amazon Linux AMI are now retired.

We highly recommend that you migrate to Amazon Linux 2023, since it's more recent than Amazon Linux 2. The Amazon Linux 2 operating system will reach end of support before Amazon Linux 2023 does, so you'll benefit from a longer time frame of support if you migrate to Amazon Linux 2023.

It's worthwhile to note that there is a high degree of compatibility between the Elastic Beanstalk Amazon Linux 2 and Amazon Linux 2023 platforms. Although some areas do have differences: the Instance Metadata Service Version 1 (IMDSv1) option default, support for the pkg-repo instance tool, and some Apache HTTPd configuration. For more information, see Amazon Linux 2023

Differences and compatibility

The AL2023/AL2 based platform branches aren't guaranteed to be backward compatible with your existing application. It's also important to be aware that even if your application code successfully deploys to the new platform version, it might behave or perform differently due to operating system and run time differences.

Although Amazon Linux AMI and AL2023/AL2 share the same Linux kernel, they differ in the following aspects: their initialization system, the `libc` versions, the compiler tool chain, and various packages. For more information, see Amazon Linux 2 FAQs.

The Elastic Beanstalk service has also updated platform specific versions of runtime, build tools, and other dependencies.

Therefore we recommend that you take your time, test your application thoroughly in a development environment, and make any necessary adjustments.

General migration process

When you're ready to go to production, Elastic Beanstalk requires a blue/green deployment to perform the upgrade. The following are the general best practice steps that we recommend for migration with a blue/green deployment procedure.

Preparing to test for your migration

Before you deploy your application and start testing, review the information in Considerations for all Linux platforms, which follows later in this topic. Also, review the information that applies to your platform in the Platform specific considerations section that follows. Make a note of the specific information from this content that applies or may apply to your application and configuration set up.

High level migration steps

1. Create a new environment that's based on an AL2 or AL2023 platform branch. We recommend that you migrate to an AL2023 platform branch.
2. Deploy your application to the target AL2023/AL2 environment.

   Your existing production environment will remain active and unaffected, while you iterate through testing and making adjustments to the new environment.
3. Test your application thoroughly in the new environment.
4. When your destination AL2023/AL2 environment is ready to go to production, swap the CNAMEs of the two environments to redirect traffic to the new environment.

More detailed migration steps and best practices

For a more detailed blue/green deployment procedure, see Blue/Green deployments with Elastic Beanstalk.

For more specific guidance and detailed best practice steps, see Blue/Green method.

More references to help plan your migration

The following references can offer additional information to plan your migration.

- Comparing Amazon Linux 2 and Amazon Linux 2023 *Amazon Linux 2023 User Guide*.
- What is Amazon Linux 2023? in the *Amazon Linux 2023 User Guide*
- Elastic Beanstalk supported platforms in *AWS Elastic Beanstalk Platforms*
- Retired platform branch history
- the section called "Linux platforms"
- Platform retirement FAQ

Considerations for all Linux platforms

The following table discusses considerations you should be aware of when planning an application migration to AL2023/AL2. These considerations apply to any of the Elastic Beanstalk Linux platforms, regardless of specific programming languages or application servers.

Area	Changes and information
Configuration Files	On AL2023/AL2 platforms, you can use configuration files as before, and all sections work the same way. However, specific settings might not work the same as they did on previous Amazon Linux AMI platforms. For example: • Some software packages that you install using a configuration file might not be available on AL2023/AL2, or their names might have changed. • Some platform specific configuration options have moved from their platform specific namespaces to different, platform agnostic namespaces. • Proxy configuration files provided in the `.ebextensions/nginx` directory should move to the `.platform/nginx` platform hooks directory. For details, see Reverse proxy configuration. We recommend using platform hooks to run custom code on your environment instances. You can still use commands and container commands in `.ebextensions` configuration files, but they aren't as easy to work with. For example, writing command scripts inside a YAML file can be cumbersome and difficult to test. You still need to use `.ebextensions` configuration files for any script that needs a reference to an AWS CloudFormation resource.

Area	Changes and information
Platform hooks	AL2 platforms introduced a new way to extend your environment's platform by adding executable files to hook directories on the environment's instances. With previous Linux platform versions, you might have used custom platform hooks. These hooks weren't designed for managed platforms and weren't supported, but could work in useful ways in some cases. With AL2023/AL2 platform versions, custom platform hooks don't work. You should migrate any hooks to the new platform hooks. For details see Platform hooks.
Supported proxy servers	AL2023/AL2 platform versions support the same reverse proxy servers as each platform supported in its Amazon Linux AMI platform versions. All AL2023/AL 2; platform versions use nginx as their default reverse proxy server, with the exception of the ECS and Docker platforms. The Tomcat, Node.js, PHP, and Python platform also support Apache HTTPD as an alternative. All platforms enable proxy server configuration in a uniform way, as described in this section. However, configuring the proxy server is slightly different than it was on Amazon Linux AMI. These are the differences for all platforms: • **Default is nginx** – The default proxy server on all AL2023/AL2 platform versions is nginx. On Amazon Linux AMI platform versions of Tomcat, PHP, and Python, the default proxy server was Apache HTTPD. • **Consistent namespace** – All AL2023/AL2 platform versions use the `aws:elasticbeanstalk:environment:proxy` namespace to configure the proxy server. On Amazon Linux AMI platform versions this was a per-platform decision, and Node.js used a different namespace. • **Configuration file location** – You should place proxy configuration files in the `.platform/nginx` and `.platform/httpd` directories on all AL2023/AL2 platform versions. On Amazon Linux AMI platform versions these locations were `.ebextensions/nginx` and `.ebextensions/httpd` , respectively. For platform-specific proxy configuration changes, see the section called "Platform specific considerations". For information about proxy configuration on AL2023/AL2 platforms, see Reverse proxy configuration.

Area	Changes and information
Proxy Configuration Changes	There are proxy configuration changes that apply uniformly to all platforms in addition to proxy configuration changes that are specific to each platform. It's important to refer to both to accurately configure your environments. • **All platforms** – see Reverse proxy configuration • **Platform-specific** – see the section called "Platform specific considerations".
Instance profile	AL2023/AL2 platforms require an instance profile to be configured. Environment creation might temporarily succeed without one, but the environment might show errors soon after creation when actions requiring an instance profile start failing. For details, see the section called "Instance profiles".
Enhanced health	AL2023/AL2 platform versions enable enhanced health by default. This is a change if you don't use the Elastic Beanstalk console to create your environments. The console enables enhanced health by default whenever possible, regardless of platform version. For details, see the section called "Enhanced health reporting and monitoring".
Custom AMI	If your environment uses a custom AMI, create a new AMI based on AL2023/AL2 for your new environment using an Elastic Beanstalk AL2023/AL2 platform.
Custom platforms	The managed AMIs of AL2023/AL2 platform versions don't support custom platforms.

Platform specific considerations

This section discusses migration considerations specific to particular Elastic Beanstalk Linux platforms.

Docker

The Docker platform branch family based on Amazon Linux AMI (AL1) includes three platform branches. We recommend a different migration path for each.

AL1 Platform branch	Migration Path to AL2023/AL2
Multi-container Docker managed by Amazon ECS running on Amazon Linux AMI (AL1)	**ECS based Docker AL2023/AL2 platform branches** The *ECS based Docker AL2023/AL2* platform branches offer a straightforward migration path for environments running on the *Multi-container Docker AL1* platform branch. • Like the previous *Multi-container Docker AL1* branch, the AL2023/AL2 platform branches use Amazon ECS to coordinate deployment of multiple Docker containers to an Amazon ECS cluster in an Elastic Beanstalk environment. • The AL2023/AL2 platform branches support all of the features in the previous *Multi-container Docker AL1* branch. • The AL2023/AL2 platform branches also support the same `Dockerrun .aws.json` v2 file. For more information about migrating your applications running on the *Multi-container Docker Amazon Linux* platform branch to an *Amazon ECS running on AL2023/AL2* platform branch, see ???.
Docker running on Amazon Linux AMI (AL1) Preconfigured Docker (Glassfish 5.0) running Amazon Linux AMI (AL1)	**Docker Running on AL2023/AL2 platform branch** We recommend that you migrate your applications running on environments based on *Preconfigured Docker (Glassfish 5.0)* or *Docker running on Amazon Linux AMI (AL1)* to environments that are based on the *Docker Running on Amazon Linux 2* or *Docker Running on AL2023* platform branches. If your environment is based on the *Preconfigured Docker (Glassfish 5.0)* platform branch, see the section called “Tutorial - GlassFish on Docker: path to Amazon Linux 2023”. The following table lists migration information specific to the platform branch *Docker Running on AL2023/AL2*.

AL1 Platform branch	Migration Path to AL2023/AL2
	<table><tr><th>Area</th><th>Changes and information</th></tr><tr><td>Storage</td><td>Elastic Beanstalk configures Docker to use storage drivers to store Docker images and container data. On Amazon Linux AMI, Elastic Beanstalk used the Device Mapper storage driver. To improve performance, Elastic Beanstalk provisioned an extra Amazon EBS volume. On AL2023/AL2 Docker platform versions, Elastic Beanstalk uses the OverlayFS storage driver, and achieves even better performance while not requiring a separate volume anymore. With Amazon Linux AMI, if you used the `BlockDeviceMappings` option of the `aws:autoscaling:launchconfiguration` namespace to add custom storage volumes to a Docker environment, we advised you to also add the `/dev/xvdcz` Amazon EBS volume that Elastic Beanstalk provisions. Elastic Beanstalk doesn't provision this volume anymore, so you should remove it from your configuration files. For details, see the section called “Docker configuration on Amazon Linux AMI (preceding Amazon Linux 2)”.</td></tr><tr><td>Private repository authentication</td><td>When you provide a Docker-generated authentication file to connect to a private repository, you no longer need to convert it to the older format that Amazon Linux AMI Docker platform versions required. AL2023/AL2 Docker platform versions support the new format. For details, see the section called “Authenticating with image repositories”.</td></tr><tr><td>Proxy server</td><td>AL2023/AL2 Docker platform versions don't support standalone containers that don't run behind a proxy server. On Amazon Linux AMI Docker platform versions, this used to be possible through the none value of the `ProxyServer` option in the `aws:elasticbeanstalk:environment:proxy` namespace.</td></tr></table>

Go

The following table lists migration information for the AL2023/AL2 platform versions in the Go platform.

Area	Changes and information
Port passing	On AL2023/AL2 platforms, Elastic Beanstalk doesn't pass a port value to your application process through the `PORT` environment variable. You can simulate this behavior for your process by configuring a `PORT` environment property yourself. However, if you have multiple processes, and you're counting on Elastic Beanstalk passing incremental port values to your processes (5000, 5100, 5200 etc.), you should modify your implementation. For details see Reverse proxy configuration.

Amazon Corretto

The following table lists migration information for the Corretto platform branches in the Java SE platform.

Area	Changes and information
Corretto vs. OpenJDK	To implement the Java Platform, Standard Edition (Java SE), AL2023/AL2 platform branches use Amazon Corretto, an AWS distribution of the Open Java Development Kit (OpenJDK). Prior Elastic Beanstalk Java SE platform branches use the OpenJDK packages included with Amazon Linux AMI.
Build tools	AL2023/AL2 platforms have newer versions of the build tools: `gradle`, `maven`, and `ant`.
JAR file handling	On AL2023/AL2 platforms, if your source bundle (ZIP file) contains a single JAR file and no other files, Elastic Beanstalk no longer renames the JAR file to `application.jar` . Renaming occurs only if you submit a JAR file on its own, not within a ZIP file.
Port passing	On AL2023/AL2 platforms, Elastic Beanstalk doesn't pass a port value to your application process through the `PORT` environment variable. You can simulate this behavior for your process by configuring a `PORT` environment property yourself. However, if you have multiple processes, and you're counting on Elastic Beanstalk

Area	Changes and information
	passing incremental port values to your processes (5000, 5100, 5200 etc.), you should modify your implementation. For details see Reverse proxy configuration.
Java 7	Elastic Beanstalk doesn't support an AL2023/AL2 Java 7 platform branch. If you have a Java 7 application, migrate it to Corretto 8 or Corretto 11.

Tomcat

The following table lists migration information for the AL2023/AL2 platform versions in the Tomcat platform.

Area	Changes and information
Configuration options	On AL2023/AL2 platform versions, Elastic Beanstalk supports only a subset of the configuration options and option values in the `aws:elasticbeanstalk:environment:proxy` namespace. Here's the migration information for each option.

Option	Migration information
`GzipCompression`	Unsupported on AL2023/AL2 platform versions.
`ProxyServer`	AL2023/AL2 Tomcat platform versions support both the nginx and the Apache HTTPD version 2.4 proxy servers. However, Apache version 2.2 isn't supported. On Amazon Linux AMI platform versions, the default proxy was Apache 2.4. If you used the default proxy setting and added custom proxy configuration files, your proxy configuration should still work on AL2023/AL2. However, if you used the `apache/2.2` option value, you now have to migrate your proxy configuration to Apache version 2.4.

Area	Changes and information
	The `XX:MaxPermSize` option in the `aws:elasticbeanstalk:container:tomcat:jvmoptions` namespace isn't supported on AL2023/AL2 platform versions. The JVM setting to modify the size of the permanent generation applies only to Java 7 and earlier, and is therefore not applicable to AL2023/AL2 platform versions.
Application path	On AL2023/AL2 platforms, the path to the application's directory on Amazon EC2 instances of your environment is `/var/app/current`. It was `/var/lib/tomcat8/webapps` on Amazon Linux AMI platforms.

Node.js

The following table lists migration information for the AL2023/AL2 platform versions in the Node.js platform.

Area	Changes and information
Installed Node.js versions	On AL2023/AL2 platforms, Elastic Beanstalk maintains several Node.js platform branches, and only installs the latest version of the Node.js major version corresponding with the platform branch on each platform version. For example, each platform version in the Node.js 12 platform branch only has Node.js 12.x.y installed by default. On Amazon Linux AMI platform versions, we installed the multiple versions of multiple Node.js versions on each platform version, and only maintained a single platform branch. Choose the Node.js platform branch that corresponds with the Node.js major version that your application needs.
Apache HTTPD log file names	On AL2023/AL2 platforms, if you use the Apache HTTPD proxy server, the HTTPD log file names are `access_log` and `error_log`, which is consistent with all other platforms that support Apache HTTPD. On Amazon Linux AMI platform versions, these log files were named `access.log` and `error.log`, respectively. For details about log file names and locations for all platforms, see the section called "How Elastic Beanstalk sets up CloudWatch Logs".

Area	Changes and information
Configuration options	On AL2023/AL2 platforms, Elastic Beanstalk doesn't support the configuration options in the `aws:elasticbeanstalk:container:nodejs` namespace. Some of the options have alternatives. Here's the migration information for each option. **Option** | **Migration information** `NodeCommand` | Use a `Procfile` or the `scripts` keyword in a `package.json` file to specify the start script. `NodeVersion` | Use the `engines` keyword in a `package.json` file to specify the Node.js version. Be aware that you can only specify a Node.js version that correspondes with your platform branch. For example, if you're using the Node.js 12 platform branch, you can specify only a 12.x.y Node.js version. For details, see the section called "Specifying Node.js dependencies with a package.json file". `GzipCompression` | Unsupported on AL2023/AL2 platform versions. `ProxyServer` | On AL2023/AL2 Node.js platform versions, this option moved to the `aws:elasticbeanstalk:environment:proxy` namespace. You can choose between `nginx` (the default) and `apache`. AL2023/AL2 Node.js platform versions don't support standalone applications that don't run behind a proxy server. On Amazon Linux AMI Node.js platform versions, this used to be possible through the none value of the `ProxyServer` option in the `aws:elasticbeanstalk:container:nodejs` namespace. If your environment runs a standalone application, update your code to listen to the port that the proxy server (nginx or Apache) forwards traffic to. `var port = process.env.PORT || 5000;`

Area	Changes and information
	Option / **Migration information**

	`app.listen(port, function() {`
	`  console.log('Server running at http://127.0.0.1:%s',`
	` port);`
	`});`

PHP

The following table lists migration information for the AL2023/AL2 platform versions in the PHP platform.

Area	Changes and information
PHP file processing	On AL2023/AL2 platforms, PHP files are processed using PHP-FPM (a CGI process manager). On Amazon Linux AMI platforms we used mod_php (an Apache module).
Proxy server	AL2023/AL2 PHP platform versions support both the nginx and the Apache HTTPD proxy servers. The default is nginx. Amazon Linux AMI PHP platform versions supported only Apache HTTPD. If you added custom Apache configuration files, you can set the `ProxyServer` option in the `aws:elasticbeanstalk:environment:proxy` namespace to `apache`.

Python

The following table lists migration information for the AL2023/AL2 platform versions in the Python platform.

Area	Changes and information
WSGI server	On AL2023/AL2 platforms, Gunicorn is the default WSGI server. By default, Gunicorn listens on port 8000. The port might be different than what your

Area	Changes and information
	application used on the Amazon Linux AMI platform. If you're setting the `WSGIPath` option of the `aws:elasticbeanstalk:container:python` namespace, replace the value with Gunicorn's syntax. For details, see the section called "Python configuration namespaces". Alternatively, you can use a `Procfile` to specify and configure the WSGI server. For details, see the section called "Procfile".
Application path	On AL2023/AL2 platforms, the path to the application's directory on Amazon EC2 instances of your environment is `/var/app/current` . It was `/opt/python/current/app` on Amazon Linux AMI platforms.
Proxy server	AL2023/AL2 Python platform versions support both the nginx and the Apache HTTPD proxy servers. The default is nginx. Amazon Linux AMI Python platform versions supported only Apache HTTPD. If you added custom Apache configuration files, you can set the `ProxyServer` option in the `aws:elasticbeanstalk:environment:proxy` namespace to `apache`.

Ruby

The following table lists migration information for the AL2023/AL2 platform versions in the Ruby platform.

Area	Changes and information
Installed Ruby versions	On AL2023/AL2 platforms, Elastic Beanstalk only installs the latest version of a single Ruby version, corresponding with the platform branch, on each platform version. For example, each platform version in the Ruby 2.6 platform branch only has Ruby 2.6.x installed. On Amazon Linux AMI platform versions, we installed the latest versions of multiple Ruby versions, for example, 2.4.x, 2.5.x, and 2.6.x. If your application uses a Ruby version that doesn't correspond to the platform branch you're using, we recommend that you switch to a platform branch that has the correct Ruby version for your application.

Area	Changes and information
Application server	On AL2023/AL2 platforms, Elastic Beanstalk only installs the Puma application server on all Ruby platform versions. You can use a `Procfile` to start a different application server, and a `Gemfile` to install it. On the Amazon Linux AMI platform, we supported two flavors of platform branches for each Ruby version—one with the Puma application server and the other with the Passenger application server. If your application uses Passenger, you can configure your Ruby environment to install and use Passenger. For more information and examples, see the section called "The Ruby platform".

Platform retirement FAQ

Note

Elastic Beanstalk retired all platform branches based on Amazon Linux AMI (AL1) on July 18, 2022 .

The answers in this FAQ reference the following topics:

- Elastic Beanstalk platform support policy
- Retired platform branch history
- Elastic Beanstalk supported platforms in *AWS Elastic Beanstalk Platforms*
- Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2
- Amazon Linux 2 FAQs.

1. What does retirement of a platform branch mean?

Following the announced retirement date of a platform branch, you will no longer be able to create a new environment based on the retired platform branch, unless you already have an active environment based on that platform branch. For more information, see FAQ #11. Elastic Beanstalk will stop providing new maintenance updates for these platform branches. A retired platform branch isn't recommended for use in production environments. For more information, see FAQ #5.

2. Why has AWS retired the AL1-based platforms branches?

Elastic Beanstalk retires platform branches when platform components are deprecated or retired by their vendors. In this case, the Amazon Linux AMI (AL1) has ended standard support as of December 31, 2020. While Elastic Beanstalk continued to offer AL1 based platforms through 2022, we have since released AL2 and AL2023 and based platforms that have the latest features. In order for customers to continue to benefit from the latest security and features going forward, it's critical for customers to migrate to our AL2 or AL2023 based platforms.

3. Which platform branches are retired?

For a list of platform components and platform branches that have been retired, see Retired platform branch history.

4. Which platforms are currently supported?

See Elastic Beanstalk supported platforms in *AWS Elastic Beanstalk Platforms*.

5. Will Elastic Beanstalk remove or terminate any components of my environment after retirement?

Our policy for retired platform branches does not remove access to environments nor delete resources. However, an environment based on a retired platform branch can end up in an unpredictable situation, because Elastic Beanstalk isn't able to provide security updates, technical support, or hotfixes for retired platform branches due to the supplier marking their component as End of Life (EOL). For example, a detrimental and critical security vulnerability may surface in an environment running on a retired platform branch. Or an EB API action may stop working for the environment if it becomes incompatible with the Elastic Beanstalk service over time. The opportunity for these types of risks increases the longer an environment based on a retired platform branch remains active.

If your application should encounter issues while running on a retired platform branch and you're not able to migrate it to a supported platform, you'll need to consider other alternatives. Workarounds include encapsulating the application into a Docker image to run it as a Docker container. This would allow a customer to use any of our Docker solutions, such as our Elastic Beanstalk AL2023/AL2 Docker platforms, or other Docker based services such as Amazon ECS or Amazon EKS. Non-Docker alternatives include our AWS CodeDeploy service, which allows complete customization of the runtimes you desire.

6. Can I submit a request to extend the retirement date?

No. After the retirement date existing environments will continue to function. However, Elastic Beanstalk will no longer provide platform maintenance and security updates. Therefore, it's critical to migrate to AL2 or AL2023 if you are still running applications on an AL1-based platform. For more information about risks and workarounds, see FAQ #5.

7. What are the workarounds if I can't complete my AL2 or AL2023 migration in time?

Customers may continue to run the environment, although we strongly encourage you to plan to migrate all of your Elastic Beanstalk environments to a supported platform version. Doing so will minimize risk and provide continued benefit from important security, performance, and functionality enhancements offered in more recent releases. For more information about risks and workarounds, see FAQ #5.

8. What is the recommended process to migrate to AL2 or AL2023 platforms?

For comprehensive AL1 to AL2023/AL2 migration instructions, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2. This topic explains that Elastic Beanstalk requires a blue/green deployment to perform the upgrade.

9. If I have an environment running on a retired platform, what would be the impact?

An environment based on a retired platform branch can end up in an unpredictable situation, because Elastic Beanstalk isn't able to provide security updates, technical support, or hotfixes for retired platform branches due to the supplier marking their component as End of Life (EOL). For example, a detrimental and critical security vulnerability may surface in an environment running on a retired platform branch. Or an EB API action may stop working for the environment if it becomes incompatible with the Elastic Beanstalk service over time. The opportunity for these types of risks increases the longer an environment on a retired platform branch remains active. For more information, see FAQ #5.

10. What happens 90 days after the retirement date?

Our policy for retired platform branches does not remove access to environments nor delete resources. However, be aware that an environment based on a retired platform branch can end up in an unpredictable situation, because Elastic Beanstalk isn't able to provide security updates, technical support, or hotfixes for retired platform branches due to the supplier marking their component as End of Life (EOL). For example, a detrimental and critical security vulnerability may surface in an environment running on a retired platform branch. Or an EB API action may stop working for the environment if it becomes incompatible with the Elastic Beanstalk service over time. The opportunity for these types of risks increases the longer an environment on a retired platform branch remains active. For more information see FAQ #5.

11. Can I create a new environment based on a retired platform?

You can create a new environment based on a retired platform branch, if you've already used that platform branch to create an existing environment using the same account and in the same region. The retired platform branch will not be available in the Elastic Beanstalk console. However, for customers that have existing environments based on a retired platform branch, it will be available through the EB CLI, EB API, and AWS CLI. Also, existing customers can use the Clone environment and Rebuild environment consoles. However, be aware that an environment based on a retired platform branch can end up in an unpredictable situation. For more information, see FAQ #5.

12. If I have an existing environment running on a retired platform branch, until when can I create a new environment based on the retired platform branch? Can I do so using the console, CLI or API?

You can create the environment after the retirement date. However, keep in mind that a retired platform branch can end up in an unpredictable situation. The further out in time such an environment an environment is created or active, the higher the risk for the environment to encounter unexpected issues. For more information about creating a new environment, see FAQ #11.

13. Can I clone or rebuild my environment which is based on retired platform?

Yes. You can do so using the Clone environment and Rebuild environment consoles. You can also use the EB CLI, EB API, and AWS CLI. For more information about creating a new environment, see FAQ #11.

However, we strongly encourage you to plan to migrate all your Elastic Beanstalk environments to a supported platform version. Doing so will minimize risk and provide continued benefit from important security, performance, and functionality enhancements offered in more recent releases. For more information about risks and workarounds, see FAQ #5.

14. After the retirement date, what would happen to the AWS resources of my Elastic Beanstalk environment that is based on a retired platform branch? For example, if the running EC2 instance gets terminated, would Elastic Beanstalk be able to launch a new AL1 based EC2 instance to maintain capacity?

The environment’s resources would remain active and continue to function. And, yes, Elastic Beanstalk will auto scale for AL1 EC2 instances in the environment. However, Elastic Beanstalk will stop providing new platform maintenance updates to the environment, which can lead to the environment ending up in an unpredictable situation over time. For more information, see FAQ #5.

15. What are key differences between the AL2023/AL2 and Amazon Linux AMI (AL1) operating systems? How are the Elastic Beanstalk AL2023/AL2 platform branches affected?

Although Amazon Linux AMI and AL2023/AL2 share the same Linux kernel, they differ in their initialization system, `libc` versions, the compiler tool chain, and various packages. For more information, see Amazon Linux 2 FAQs.

The Elastic Beanstalk service has also updated platform specific versions of runtime, build tools, and other dependencies. The AL2023/AL2 based platform branches aren't guaranteed to be backward compatible with your existing application. Furthermore, even if your application code successfully deploys to the new platform version, it might behave or perform differently due to operating system and run time differences. For a list and description of configurations and customizations that you'll need to review and test, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

Canceling environment configuration updates and application deployments

You can cancel in-progress updates that are triggered by environment configuration changes. You can also cancel the deployment of a new application version in progress. For example, you might want to cancel an update if you decide you want to continue using the existing environment configuration instead of applying new environment configuration settings. Or, you might realize that the new application version that you are deploying has problems that will cause it to not start or not run properly. By canceling an environment update or application version update, you can avoid waiting until the update or deployment process is done before you begin a new attempt to update the environment or application version.

Note

During the cleanup phase in which old resources that are no longer needed are removed, after the last batch of instances has been updated, you can no longer cancel the update.

Elastic Beanstalk performs the rollback the same way that it performed the last successful update. For example, if you have time-based rolling updates enabled in your environment, then Elastic Beanstalk will wait the specified pause time between rolling back changes on one batch of instances before rolling back changes on the next batch. Or, if you recently turned on rolling updates, but the last time you successfully updated your environment configuration settings was without rolling updates, Elastic Beanstalk will perform the rollback on all instances simultaneously.

You cannot stop Elastic Beanstalk from rolling back to the previous environment configuration once it begins to cancel the update. The rollback process continues until all instances in the environment have the previous environment configuration or until the rollback process fails. For application version deployments, canceling the deployment simply stops the deployment; some instances will have the new application version and others will continue to run the existing application version. You can deploy the same or another application version later.

For more information about rolling updates, see Elastic Beanstalk rolling environment configuration updates. For more information about batched application version deployments, see Deployment policies and settings.

To cancel an update

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the environment overview page, choose **Actions**, and then choose **Abort current operation**.

Rebuilding Elastic Beanstalk environments

Your AWS Elastic Beanstalk environment can become unusable if you don't use Elastic Beanstalk functionality to modify or terminate the environment's underlying AWS resources. If this happens, you can **rebuild** the environment to attempt to restore it to a working state. Rebuilding an environment terminates all of its resources and replaces them with new resources with the same configuration.

You can also rebuild terminated environments within six weeks (42 days) of their termination. When you rebuild, Elastic Beanstalk attempts to create a new environment with the same name, ID, and configuration.

Rebuilding a running environment

You can rebuild an environment through the Elastic Beanstalk console or by using the `RebuildEnvironment` API.

Warning

If your environment has a coupled database, **it will be deleted as part of the rebuild**, and the new database in the rebuilt environment will not contain the previous data. If you would like to retain the database or take a snapshot, make sure you have the database deletion policy configured properly for the desired results after it's rebuilt. For more information, see Database lifecycle.

To rebuild a running environment (console)

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Rebuild environment**.
4. Choose **Rebuild**.

To rebuild a running environment with the Elastic Beanstalk API, use the `RebuildEnvironment` action with the AWS CLI or the AWS SDK.

```
$ aws elasticbeanstalk rebuild-environment --environment-id e-vdnftxubwq
```

Rebuilding a terminated environment

You can rebuild and restore a terminated environment by using the Elastic Beanstalk console, the EB CLI, or the `RebuildEnvironment` API.

Note

Unless you are using your own custom domain name with your terminated environment, the environment uses a subdomain of elasticbeanstalk.com. These subdomains are shared within an Elastic Beanstalk region. Therefore, they can be used by any environment created by any customer in the same region. While your environment was terminated, another environment could use its subdomain. In this case, the rebuild would fail.

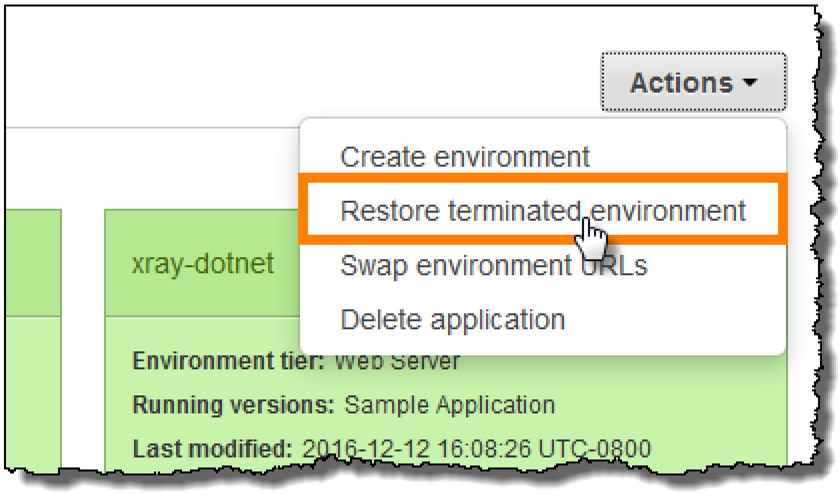
You can avoid this issue by using a custom domain. See Your Elastic Beanstalk environment's Domain name for details.

Recently terminated environments appear in the application overview for up to an hour. During this time, you can view events for the environment in its dashboard, and use the **Restore environment** action to rebuild it.

To rebuild an environment that is no longer visible, use the **Restore terminated environment** option from the application page.

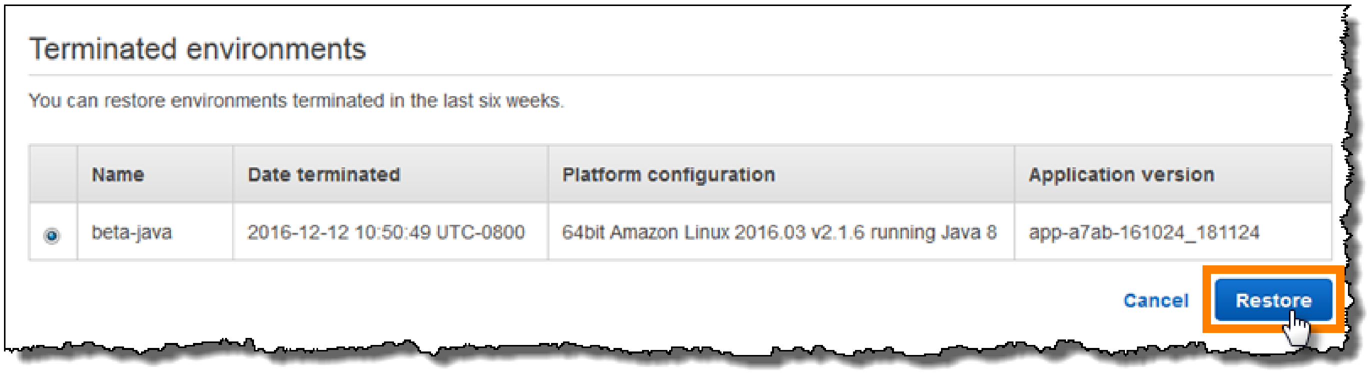
To rebuild a terminated environment (console)

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.
3. Choose **Actions**, and then choose **Restore terminated environment**.



Environment tier: Web Server
Running versions: Sample Application
Last modified: 2016-12-12 16:08:26 UTC-0800

4. Choose a terminated environment.
5. Choose **Restore**.



Terminated environments

You can restore environments terminated in the last six weeks.

	Name	Date terminated	Platform configuration	Application version
●	beta-java	2016-12-12 10:50:49 UTC-0800	64bit Amazon Linux 2016.03 v2.1.6 running Java 8	app-a7ab-161024_181124

Elastic Beanstalk attempts to create a new environment with the same name, ID, and configuration. If an environment with the same name or URL exists when you attempt to rebuild, the rebuild fails. Deleting the application version that was deployed to the environment will also cause the rebuild to fail.

If you use the EB CLI to manage your environment, use the **eb restore** command to rebuild a terminated environment.

```
$ eb restore e-vdnftxubwq
```

See eb restore for more information.

To rebuild a terminated environment with the Elastic Beanstalk API, use the `RebuildEnvironment` action with the AWS CLI or the AWS SDK.

```
$ aws elasticbeanstalk rebuild-environment --environment-id e-vdnftxubwq
```

Environment types

In AWS Elastic Beanstalk, you can create a load-balanced, scalable environment or a single-instance environment. The type of environment that you require depends on the application that you deploy. For example, you can develop and test an application in a single-instance environment to save costs and then upgrade that environment to a load-balanced, scalable environment when the application is ready for production.

Note

A worker environment tier for a web application that processes background tasks doesn't include a load balancer. However, a worker environment does effectively scale out by adding instances to the Auto Scaling group to process data from the Amazon SQS queue when the load necessitates it.

Load-balanced, scalable environment

A load-balanced and scalable environment uses the Elastic Load Balancing and Amazon EC2 Auto Scaling services to provision the Amazon EC2 instances that are required for your deployed application. Amazon EC2 Auto Scaling automatically starts additional instances to accommodate increasing load on your application. If the load on your application decreases, Amazon EC2 Auto Scaling stops instances but always leaves your specified minimum number of instances running. If your application requires scalability with the option of running in multiple Availability Zones, use a load-balanced, scalable environment. If you're not sure which environment type to select, you can pick one and, if required, switch the environment type later.

Single-instance environment

A single-instance environment contains one Amazon EC2 instance with an Elastic IP address. A single-instance environment doesn't have a load balancer, which can help you reduce costs compared to a load-balanced, scalable environment. Although a single-instance environment does use the Amazon EC2 Auto Scaling service, settings for the minimum number of instances, maximum number of instances, and desired capacity are all set to 1. Consequently, new instances are not started to accommodate increasing load on your application.

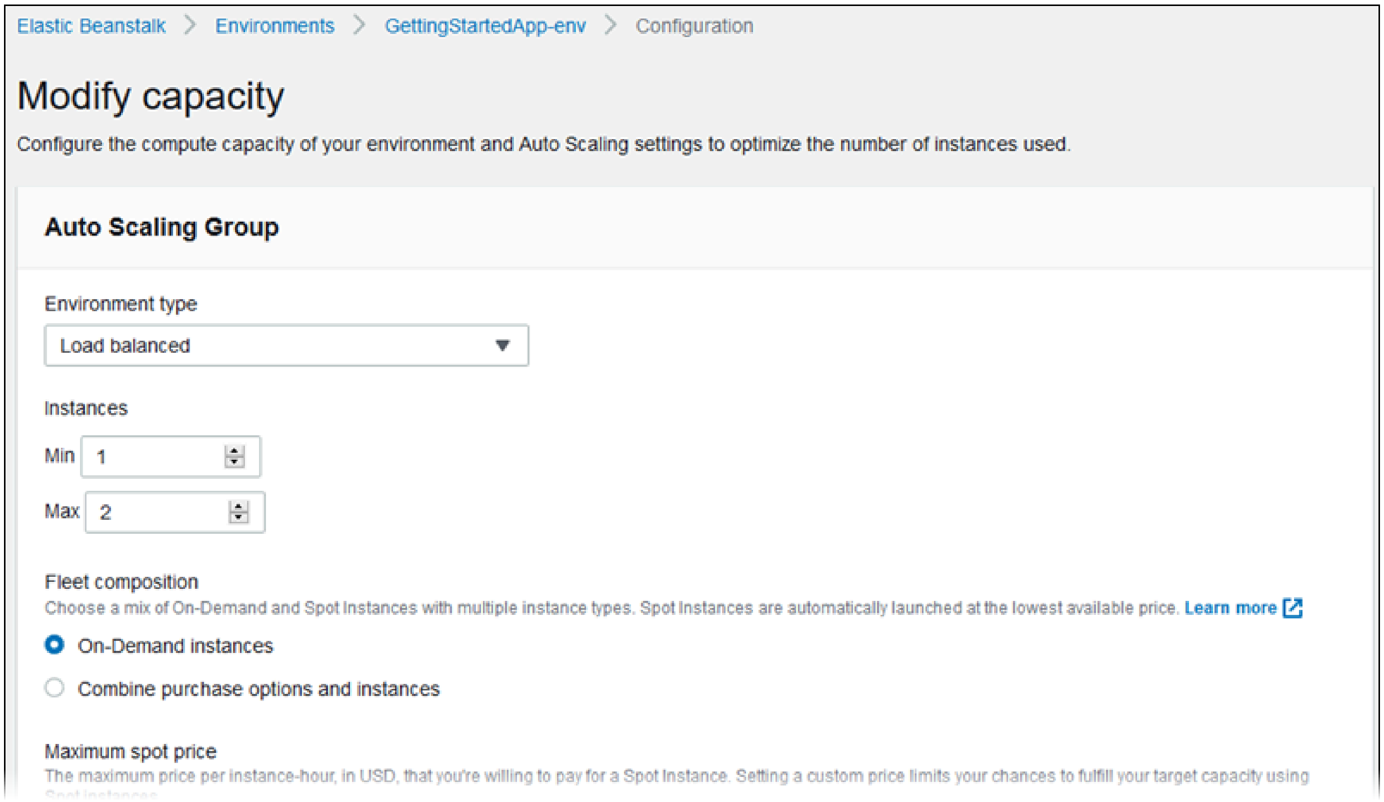
Use a single-instance environment if you expect your production application to have low traffic or if you are doing remote development. If you're not sure which environment type to select, you can pick one and, if required, you can switch the environment type later. For more information, see Changing environment type.

Changing environment type

You can change your environment type to a single-instance or load-balanced, scalable environment by editing your environment's configuration. In some cases, you might want to change your environment type from one type to another. For example, let's say that you developed and tested an application in a single-instance environment to save costs. When your application is ready for production, you can change the environment type to a load-balanced, scalable environment so that it can scale to meet the demands of your customers.

To change an environment's type

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** category, choose **Edit**.
5. From the **Environment Type** list, select the type of environment that you want.



Elastic Beanstalk > Environments > GettingStartedApp-env > Configuration

Modify capacity

Configure the compute capacity of your environment and Auto Scaling settings to optimize the number of instances used.

Auto Scaling Group

Environment type: Load balanced

Instances

Min: 1

Max: 2

Fleet composition

Choose a mix of On-Demand and Spot Instances with multiple instance types. Spot Instances are automatically launched at the lowest available price. Learn more

[✓ On-Demand instances] Combine purchase options and instances

Maximum spot price

The maximum price per instance-hour, in USD, that you're willing to pay for a Spot Instance. Setting a custom price limits your chances to fulfill your target capacity using

6. Choose **Save**.

   It can take several minutes for the environment to update while Elastic Beanstalk provisions AWS resources.

If your environment is in a VPC, select subnets to place Elastic Load Balancing and Amazon EC2 instances in. Each Availability Zone that your application runs in must have both. See Using Elastic Beanstalk with Amazon VPC for details.

Elastic Beanstalk worker environments

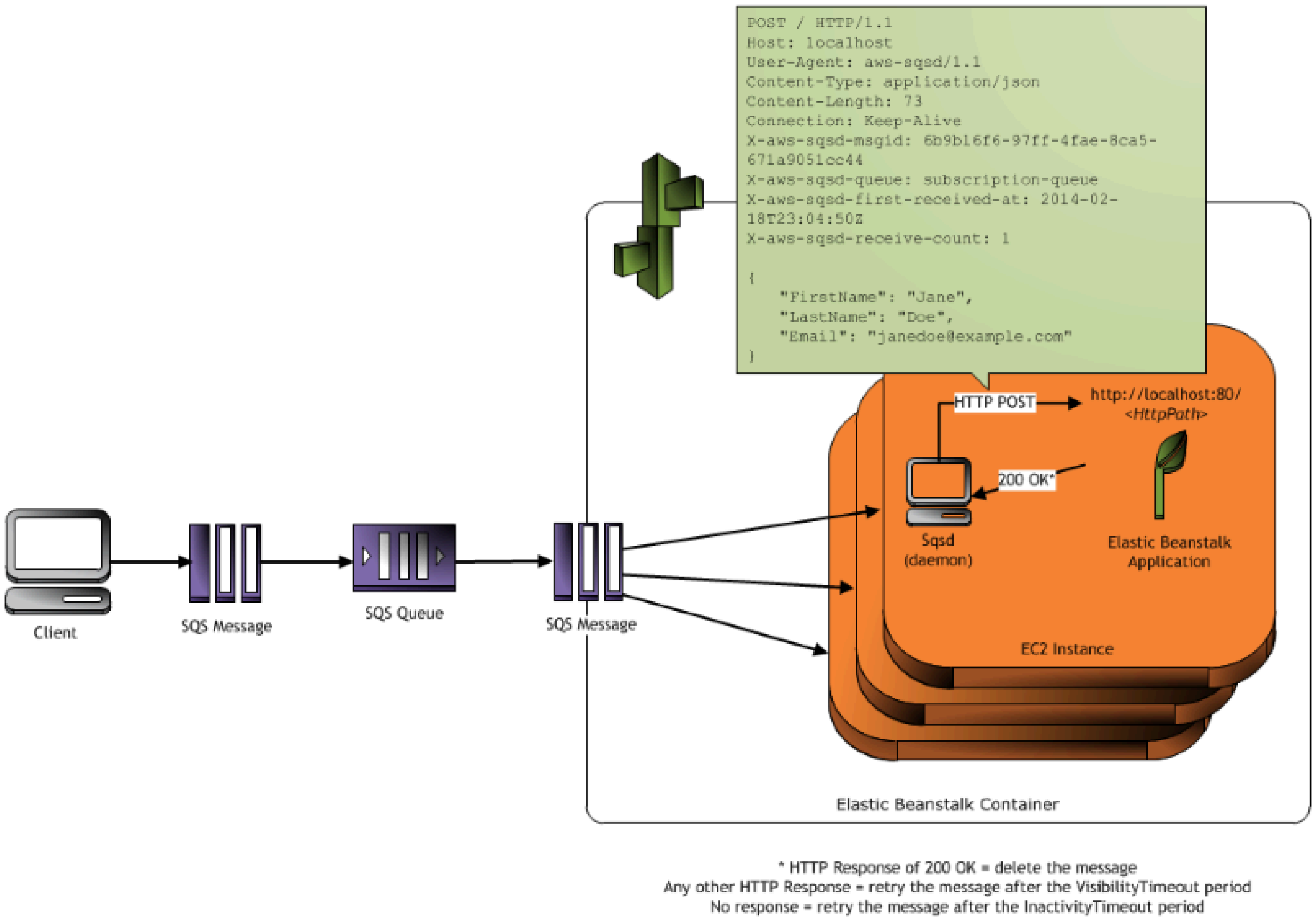
If your AWS Elastic Beanstalk application performs operations or workflows that take a long time to complete, you can offload those tasks to a dedicated *worker environment*. Decoupling your web application front end from a process that performs blocking operations is a common way to ensure that your application stays responsive under load.

A long-running task is anything that substantially increases the time it takes to complete a request, such as processing images or videos, sending email, or generating a ZIP archive. These operations can take only a second or two to complete, but a delay of a few seconds is a lot for a web request that would otherwise complete in less than 500 ms.

One option is to spawn a worker process locally, return success, and process the task asynchronously. This works if your instance can keep up with all of the tasks sent to it. Under high load, however, an instance can become overwhelmed with background tasks and become unresponsive to higher priority requests. If individual users can generate multiple tasks, the increase in load might not correspond to an increase in users, making it hard to scale out your web server tier effectively.

To avoid running long-running tasks locally, you can use the AWS SDK for your programming language to send them to an Amazon Simple Queue Service (Amazon SQS) queue, and run the process that performs them on a separate set of instances. You then design these worker instances to take items from the queue only when they have capacity to run them, preventing them from becoming overwhelmed.

Elastic Beanstalk worker environments simplify this process by managing the Amazon SQS queue and running a daemon process on each instance that reads from the queue for you. When the daemon pulls an item from the queue, it sends an HTTP POST request locally to `http://localhost/` on port 80 with the contents of the queue message in the body. All that your application needs to do is perform the long-running task in response to the POST. You can configure the daemon to post to a different path, use a MIME type other than application/JSON, connect to an existing queue, or customize connections (maximum concurrent requests), timeouts, and retries.



```
POST / HTTP/1.1
Host: localhost
User-Agent: aws-sqsd/1.1
Content-Type: application/json
Content-Length: 73
Connection: Keep-Alive
X-aws-sqsd-msgid: 6b9b16f6-97ff-4fae-8ca5-
671a9051cc44
X-aws-sqsd-queue: subscription-queue
X-aws-sqsd-first-received-at: 2014-02-
18T23:04:50Z
X-aws-sqsd-receive-count: 1

{
   "FirstName": "Jane",
   "LastName": "Doe",
   "Email": "janedoe@example.com"
}
```

Client
SQS Message
SQS Queue
SQS Message
HTTP POST
http://localhost:80/<HttpPath>
200 OK*
Sqsd (daemon)
Elastic Beanstalk Application
EC2 Instance
Elastic Beanstalk Container

* HTTP Response of 200 OK = delete the message
Any other HTTP Response = retry the message after the VisibilityTimeout period
No response = retry the message after the InactivityTimeout period

With periodic tasks, you can also configure the worker daemon to queue messages based on a cron schedule. Each periodic task can POST to a different path. Enable periodic tasks by including a YAML file in your source code that defines the schedule and path for each task.

Note

The .NET on Windows Server platform doesn't support worker environments.

Sections

- The worker environment SQS daemon
- Dead-letter queues
- Periodic tasks
- Use Amazon CloudWatch for automatic scaling in worker environment tiers

- Configuring worker environments

The worker environment SQS daemon

Worker environments run a daemon process provided by Elastic Beanstalk. This daemon is updated regularly to add features and fix bugs. To get the latest version of the daemon, update to the latest platform version.

When the application in the worker environment returns a `200 OK` response to acknowledge that it has received and successfully processed the request, the daemon sends a `DeleteMessage` call to the Amazon SQS queue to delete the message from the queue. If the application returns any response other than `200 OK`, Elastic Beanstalk waits to put the message back in the queue after the configured `ErrorVisibilityTimeout` period. If there is no response, Elastic Beanstalk waits to put the message back in the queue after the `InactivityTimeout` period so that the message is available for another attempt at processing.

Note

The properties of Amazon SQS queues (message order, at-least-once delivery, and message sampling) can affect how you design a web application for a worker environment. For more information, see Properties of Distributed Queues in the Amazon Simple Queue Service Developer Guide.

Amazon SQS automatically deletes messages that have been in a queue for longer than the configured `RetentionPeriod`.

The daemon sets the following HTTP headers.

HTTP headers

Name	Value

`User-Agent`	`aws-sqsd`
	`aws-sqsd/1.1 1`

HTTP headers

`X-Aws-Sqsd-Msgid`	SQS message ID, used to detect message storms (an unusually high number of new messages).
`X-Aws-Sqsd-Queue`	Name of the SQS queue.
`X-Aws-Sqsd-First-Received-At`	UTC time, in ISO 8601 format, when the message was first received.
`X-Aws-Sqsd-Receive-Count`	SQS message receive count.
`X-Aws-Sqsd-Attr-` *`message-attribute-name`*	Custom message attributes assigned to the message being processed. The `message-attribute-name` is the actual message attribute name. All string and number message attributes are added to the header. Binary attributes are discarded and not included in the header.
`Content-Type`	Mime type configuration; by default, `application/json` .

Dead-letter queues

Elastic Beanstalk worker environments support Amazon Simple Queue Service (Amazon SQS) dead-letter queues. A dead-letter queue is a queue where other (source) queues can send messages that for some reason could not be successfully processed. A primary benefit of using a dead-letter queue is the ability to sideline and isolate the unsuccessfully processed messages. You can then analyze any messages sent to the dead-letter queue to try to determine why they were not successfully processed.

If you specify an autogenerated Amazon SQS queue at the time you create your worker environment tier, a dead-letter queue is enabled by default for a worker environment. If you choose an existing SQS queue for your worker environment, you must use SQS to configure a dead-letter queue independently. For information about how to use SQS to configure a dead-letter queue, see Using Amazon SQS Dead Letter Queues.

You cannot disable dead-letter queues. Messages that cannot be delivered are always eventually sent to a dead-letter queue. You can, however, effectively disable this feature by setting the `MaxRetries` option to the maximum valid value of 100.

If a dead-letter queue isn't configured for your worker environment's Amazon SQS queue, Amazon SQS keeps messages on the queue until the retention period expires. For details about configuring the retention period, see the section called “Configuring worker environments”.

Note

The Elastic Beanstalk `MaxRetries` option is equivalent to the SQS `MaxReceiveCount` option. If your worker environment doesn't use an autogenerated SQS queue, use the `MaxReceiveCount` option in SQS to effectively disable your dead-letter queue. For more information, see Using Amazon SQS Dead Letter Queues.

For more information about the lifecycle of an SQS message, go to Message Lifecycle.

Periodic tasks

You can define periodic tasks in a file named `cron.yaml` in your source bundle to add jobs to your worker environment's queue automatically at a regular interval.

For example, the following `cron.yaml` file creates two periodic tasks. The first one runs every 12 hours and the second one runs at 11 PM UTC every day.

Example cron.yaml

```
version: 1
cron:
 - name: "backup-job"
   url: "/backup"
   schedule: "0 */12 * * *"
 - name: "audit"
   url: "/audit"
   schedule: "0 23 * * *"
```

The **name** must be unique for each task. The URL is the path to which the POST request is sent to trigger the job. The schedule is a CRON expression that determines when the task runs.

When a task runs, the daemon posts a message to the environment's SQS queue with a header indicating the job that needs to be performed. Any instance in the environment can pick up the message and process the job.

Note

If you configure your worker environment with an existing SQS queue and choose an Amazon SQS FIFO queue, periodic tasks aren't supported.

Elastic Beanstalk uses leader election to determine which instance in your worker environment queues the periodic task. Each instance attempts to become leader by writing to an Amazon DynamoDB table. The first instance that succeeds is the leader, and must continue to write to the table to maintain leader status. If the leader goes out of service, another instance quickly takes its place.

For periodic tasks, the worker daemon sets the following additional headers.

HTTP headers

Name	Value
`X-Aws-Sqsd-Taskname`	For periodic tasks, the name of the task to perform.
`X-Aws-Sqsd-Scheduled-At`	Time at which the periodic task was scheduled
`X-Aws-Sqsd-Sender-Id`	AWS account number of the sender of the message

Use Amazon CloudWatch for automatic scaling in worker environment tiers

Together, Amazon EC2 Auto Scaling and CloudWatch monitor the CPU utilization of the running instances in the worker environment. How you configure the automatic scaling limit for CPU capacity determines how many instances the Auto Scaling group runs to appropriately manage the throughput of messages in the Amazon SQS queue. Each EC2 instance publishes its CPU utilization metrics to CloudWatch. Amazon EC2 Auto Scaling retrieves from CloudWatch the average CPU usage across all instances in the worker environment. You configure the upper and lower threshold as well as how many instances to add or terminate according to CPU capacity. When Amazon EC2 Auto Scaling detects that you have reached the specified upper threshold on CPU capacity, Elastic Beanstalk creates new instances in the worker environment. The instances are deleted when the CPU load drops back below the threshold.

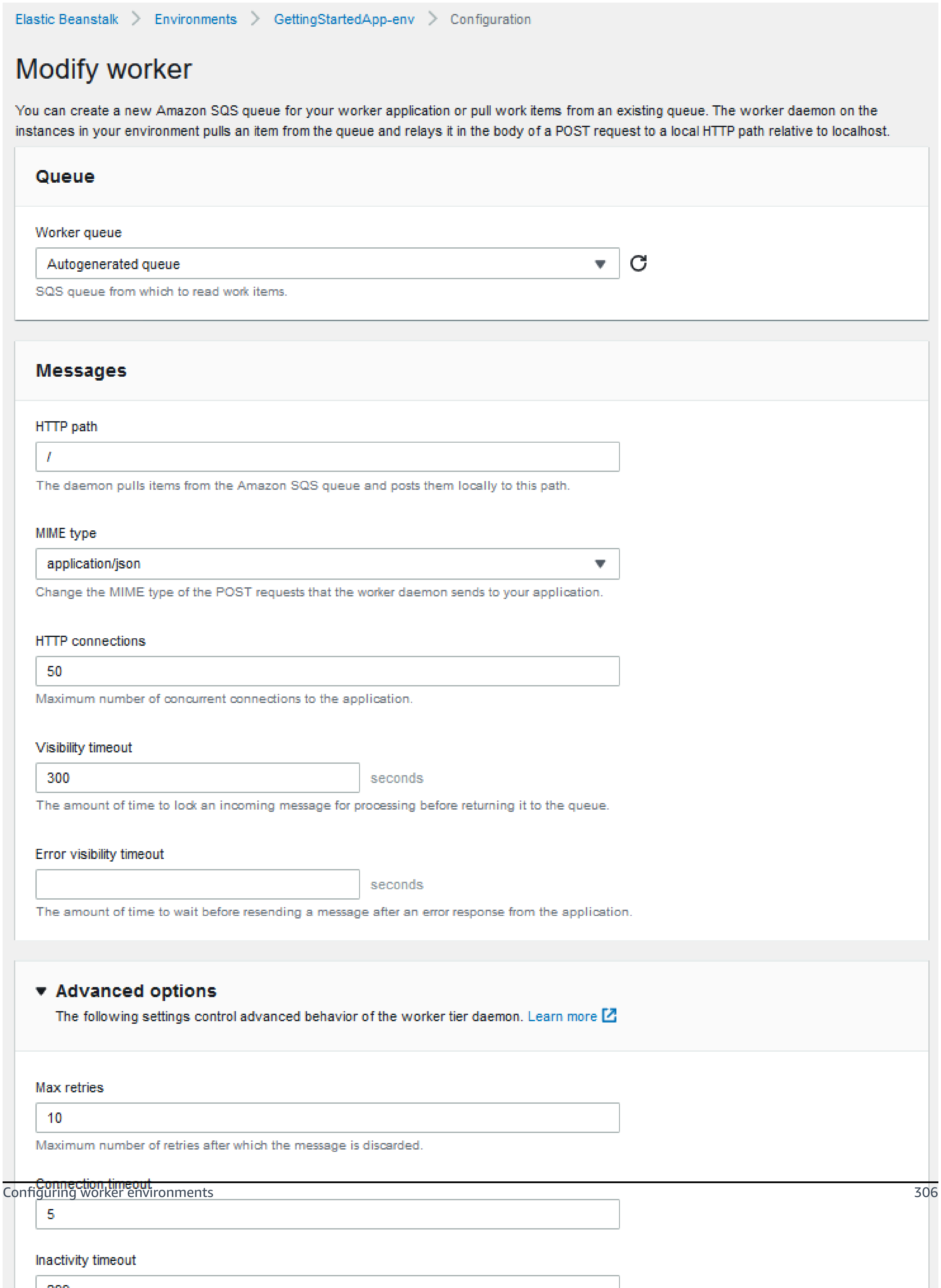
Note

Messages that have not been processed at the time an instance is terminated are returned to the queue where they can be processed by another daemon on an instance that is still running.

You can also set other CloudWatch alarms, as needed, by using the Elastic Beanstalk console, CLI, or the options file. For more information, see Using Elastic Beanstalk with Amazon CloudWatch and Create an Auto Scaling group with Step Scaling Policies.

Configuring worker environments

You can manage a worker environment's configuration by editing the **Worker** category on the **Configuration** page in the environment management console.



Elastic Beanstalk > Environments > GettingStartedApp-env > Configuration

Modify worker

You can create a new Amazon SQS queue for your worker application or pull work items from an existing queue. The worker daemon on the instances in your environment pulls an item from the queue and relays it in the body of a POST request to a local HTTP path relative to localhost.

Queue

Worker queue: Autogenerated queue
SQS queue from which to read work items.

Messages

HTTP path: /
The daemon pulls items from the Amazon SQS queue and posts them locally to this path.

MIME type: application/json
Change the MIME type of the POST requests that the worker daemon sends to your application.

HTTP connections: 50
Maximum number of concurrent connections to the application.

Visibility timeout: 300 seconds
The amount of time to lock an incoming message for processing before returning it to the queue.

Error visibility timeout: seconds
The amount of time to wait before resending a message after an error response from the application.

Advanced options

The following settings control advanced behavior of the worker tier daemon. Learn more

Max retries: 10
Maximum number of retries after which the message is discarded.

Connection timeout: 5

Inactivity timeout:

Note

You can configure the URL path for posting worker queue messages, but you can't configure the IP port. Elastic Beanstalk always posts worker queue messages on port 80. The worker environment application or its proxy must listen to port 80.

To configure the worker daemon

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Worker** configuration category, choose **Edit**.

The **Modify worker** configuration page has the following options.

In the **Queue** section:

- **Worker queue** – Specify the Amazon SQS queue from which the daemon reads. If you have one, you can choose an existing queue. If you choose **Autogenerated queue**, Elastic Beanstalk creates a new Amazon SQS queue and a corresponding **Worker queue URL**.

  Note

  When you choose **Autogenerated queue**, the queue that Elastic Beanstalk creates is a standard Amazon SQS queue. When you choose an existing queue, you can provide either a standard or a FIFO Amazon SQS queue. Be aware that if you provide a FIFO queue, periodic tasks aren't supported.

- **Worker queue URL** – If you choose an existing **Worker queue**, this setting displays the URL associated with that Amazon SQS queue.

In the **Messages** section:

- **HTTP path** – Specify the relative path to the application that will receive the data from the Amazon SQS queue. The data is inserted into the message body of an HTTP POST message. The default value is `/`.
- **MIME type** – Indicate the MIME type that the HTTP POST message uses. The default value is `application/json`. However, any value is valid because you can create and then specify your own MIME type.
- **HTTP connections** – Specify the maximum number of concurrent connections that the daemon can make to any application within an Amazon EC2 instance. The default is **`50`**. You can specify **`1`** to **`100`**.
- **Visibility timeout** – Indicate the amount of time, in seconds, an incoming message from the Amazon SQS queue is locked for processing. After the configured amount of time has passed, the message is again made visible in the queue for another daemon to read. Choose a value that is longer than you expect your application requires to process messages, up to **`43200`** seconds.
- **Error visibility timeout** – Indicate the amount of time, in seconds, that elapses before Elastic Beanstalk returns a message to the Amazon SQS queue after an attempt to process it fails with an explicit error. You can specify **`0`** to **`43200`** seconds.

In the **Advanced options** section:

- **Max retries** – Specify the maximum number of times Elastic Beanstalk attempts to send the message to the Amazon SQS queue before moving the message to the dead-letter queue. The default value is **`10`**. You can specify **`1`** to **`100`**.

  Note

  The **Max retries** option only applies to Amazon SQS queues that are configured with a dead-letter queue. For any Amazon SQS queues that aren't configured with a dead-letter queue, Amazon SQS retains messages in the queue and processes them until the period specified by the **Retention period** option expires.

- **Connection timeout** – Indicate the amount of time, in seconds, to wait for successful connections to an application. The default value is **`5`**. You can specify **`1`** to **`60`** seconds.
- **Inactivity timeout** – Indicate the amount of time, in seconds, to wait for a response on an existing connection to an application. The default value is **`180`**. You can specify **`1`** to **`36000`** seconds.

- **Retention period** – Indicate the amount of time, in seconds, a message is valid and is actively processed. The default value is **`345600`**. You can specify **`60`** to **`1209600`** seconds.

If you use an existing Amazon SQS queue, the settings that you configure when you create a worker environment can conflict with settings you configured directly in Amazon SQS. For example, if you configure a worker environment with a `RetentionPeriod` value that is higher than the `MessageRetentionPeriod` value you set in Amazon SQS, Amazon SQS deletes the message when it exceeds the `MessageRetentionPeriod`.

Conversely, if the `RetentionPeriod` value you configure in the worker environment settings is lower than the `MessageRetentionPeriod` value you set in Amazon SQS, the daemon deletes the message before Amazon SQS can. For `VisibilityTimeout`, the value that you configure for the daemon in the worker environment settings overrides the Amazon SQS `VisibilityTimeout` setting. Ensure that messages are deleted appropriately by comparing your Elastic Beanstalk settings to your Amazon SQS settings.

Creating links between Elastic Beanstalk environments

As your application grows in size and complexity, you may want to split it into components that have different development and operational lifecycles. By running smaller services that interact with each other over a well defined interface, teams can work independently and deployments can be lower risk. AWS Elastic Beanstalk lets you link your environments to share information between components that depend on one another.

Note

Elastic Beanstalk currently supports environment links for all platforms except Multicontainer Docker.

With environment links, you can specify the connections between your application's component environments as named references. When you create an environment that defines a link, Elastic Beanstalk sets an environment variable with the same name as the link. The value of the variable is the endpoint that you can use to connect to the other component, which can be a web server or worker environment.

For example, if your application consists of a frontend that collects email addresses and a worker that sends a welcome email to the email addresses collected by the frontend, you can create a link to the worker in your frontend and have the frontend automatically discover the endpoint (queue URL) for your worker.

Define links to other environments in an environment manifest, a YAML formatted file named `env.yaml` in the root of your application source. The following manifest defines a link to an environment named worker:

`~/workspace/my-app/frontend/env.yaml`

```
AWSConfigurationTemplateVersion: 1.1.0.0
EnvironmentLinks:
  "WORKERQUEUE": "worker"
```

When you create an environment with an application version that includes the above environment manifest, Elastic Beanstalk looks for an environment named `worker` that belongs to the same application. If that environment exists, Elastic Beanstalk creates an environment property named `WORKERQUEUE`. The value of `WORKERQUEUE` is the Amazon SQS queue URL. The frontend application can read this property in the same manner as an environment variable. See Environment manifest (`env.yaml`) for details.

To use environment links, add an environment manifest to your application source and upload it with the EB CLI, AWS CLI or an SDK. If you use the AWS CLI or an SDK, set the `process` flag when you call `CreateApplicationVersion`:

```
$ aws elasticbeanstalk create-application-version --process --application-name
 my-app --version-label frontend-v1 --source-bundle S3Bucket="amzn-s3-demo-
bucket",S3Key="front-v1.zip"
```

This option tells Elastic Beanstalk to validate the environment manifest and configuration files in your source bundle when you create the application version. The EB CLI sets this flag automatically when you have an environment manifest in your project directory.

Create your environments normally using any client. When you need to terminate environments, terminate the environment with the link first. If an environment is linked to by another environment, Elastic Beanstalk will prevent the linked environment from being terminated. To override this protection, use the `ForceTerminate` flag. This parameter is available in the AWS CLI as `--force-terminate`:

```
$ aws elasticbeanstalk terminate-environment --force-terminate --environment-name
 worker
```

Recovering your Elastic Beanstalk environment from an invalid state

This topic provides some background information and resources that explain how to troubleshoot an Elastic Beanstalk environment in an invalid state.

Addressing the error

Standard operations on an environment in an invalid state will not complete successfully. The failed operation will return an error that includes the following text:

```
The stack stack_id associated with environment environment-ID is in stack-status state.
```

To troubleshoot and resolve this error, see the Knowledge Center article Why is my Elastic Beanstalk environment in the invalid state?.

Note

Prior to December 16, 2024, the failing operation returned the following error instead: `Environment is in an invalid state for this operation. Must be ready.` In this case you had to contact AWS Support to reset the environment status after you completed the corrective actions.

Today you must still resolve the stack issues following the instructions in the referenced Knowledge Center article. However, once you successfully complete the corrective actions, Elastic Beanstalk automatically updates the environment's status from invalid to available, and you can resume the standard operations on your environment without further delay.

Why the error occurs

When you deploy an application in Elastic Beanstalk, the service creates an underlying AWS CloudFormation stack. Elastic Beanstalk calls the AWS CloudFormation service to launch the resources in your environment and propagate configuration changes.

If Elastic Beanstalk performs an operation on an environment without having access to a required resource, the environment's underlying CloudFormation stack can enter a failed state. Other issues can also lead to this state, although permission issues are the primary cause. As a result of the stack's failed state, AWS CloudFormation blocks Elastic Beanstalk operation requests from performing further stack updates, causing the failure of Elastic Beanstalk operations, such as UpdateEnvironment and RetrieveEnvironmentInfo.

At this point you must first correct the root cause of the underlying issue to remedy the CloudFormation stack. The Elastic Beanstalk service then detects the CloudFormation stack status change and follows through to reset your environment to an available status. At this point further operations can complete successfully.

Permission issues typically cause this effect on the CloudFormation stack and the Elastic Beanstalk environment, although out-of-band changes can also cause issues.

Important

To avoid disruption to your environment, we strongly recommend that you only initiate operations to manage and configure your environment from the Elastic Beanstalk service. Modification of resources by using the console, CLI commands, or SDK of a service other than Elastic Beanstalk is an out-of-band change, which causes *resource drift*. Resource drift affects the status of the CloudFormation stack, which in turn causes the Elastic Beanstalk environment to enter into an invalid state.
For more information about resource drift, see What is drift? in the *AWS CloudFormation User Guide*.

Configuring Elastic Beanstalk environments

This topic focuses on the configuration options available in the Elastic Beanstalk console. AWS Elastic Beanstalk provides a wide range of options for customizing the resources in your environment, along with Elastic Beanstalk behavior and platform settings.

The following topics show how to configure your environment in the console. They also describe the underlying namespaces that correspond to the console options for use with configuration files or API configuration options. To learn about advanced configuration methods, see *Configuring environments (advanced)*.

Topics

Provisioned resources

When you create a web server environment, Elastic Beanstalk creates multiple resources to support the operation of your application. This chapter describes how to customize these resources for your Elastic Beanstalk environment.

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or NGINX as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name** – A domain name that routes to your web app in the form *`subdomain`.`region`.elasticbeanstalk.com*.

Domain security

To augment the security of your Elastic Beanstalk applications, the *elasticbeanstalk.com* domain is registered in the Public Suffix List (PSL).
If you ever need to set sensitive cookies in the default domain name for your Elastic Beanstalk applications, we recommend that you use cookies with a `__Host-` prefix for increased security. This practice defends your domain against cross-site request forgery

attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

Environment configuration using the Elastic Beanstalk console

This topic outlines the configuration options available through the Elastic Beanstalk console and explains how to navigate the configuration pages.

To view a summary of your environment's configuration

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.

Configuration page

The **Configuration** overview page shows a set of configuration categories. Each configuration category groups a set of related options.

Service access

The options in this category select the *service role* and *EC2 instance profile* that Elastic Beanstalk uses to manage your environment. Optionally choose an EC2 key pair to securely log in to your EC2 instances.

Networking and database

The options in this category configure VPC settings, and subnets for the environment's EC2 instances and load balancer. They also provide the option to set up an Amazon RDS database that's integrated with your environment.

Instance traffic and scaling

These options customize the capacity, scaling, and load balancing for the environment’s EC2 instances. You can also configure Elastic Load Balancing to capture logs with detailed information about requests sent to the load balancer.

The following options for your EC2 instances are also available for configuration:

- Root volume type, size, input/output operation rate (IOPS), and throughput.
- Enabling instance metadata service (IMDS).
- Selection of EC2 security groups to control instance traffic.
- CloudWatch metrics monitoring interval.
- Time interval for metrics logging.

Updates, monitoring, and logging

This category configures the following options:

- Environment health reporting, including the option to select enhanced health reporting.
- Managed platform updates that define when and how Elastic Beanstalk deploys changes to the environment.
- Enabling of the X-Ray service to collect data about your application's behavior to identify issues and optimization opportunities.
- Platform specific options, including the proxy server and OS environment properties.

Navigating the configuration page

Choose **Edit** in a configuration category to launch the associated configuration page, where you can see full option values and make changes.

Navigation in a configuration category

Navigate in a configuration category page with any of the following actions:

- **Cancel –** Go back to the **Configuration** overview page without applying your configuration changes. When you choose **Cancel**, the console loses any pending changes you made on any configuration category.

  You can also cancel your configuration changes by choosing another item on the left navigation page, like **Events** or **Logs**.
- **Continue –** Go back to the **Configuration** overview page. You can then continue making changes or apply pending ones.
- **Apply –** Apply the changes you made in any of the configuration categories to your environment. In some cases you're prompted to confirm a consequence of one of your configuration decisions.

Navigation on the Configuration overview page

Choose **Edit** in a configuration category to launch a related configuration page, where you can see full option values and make changes. When you're done viewing and modifying options, you can choose one of the following actions from the **Configuration** overview page:

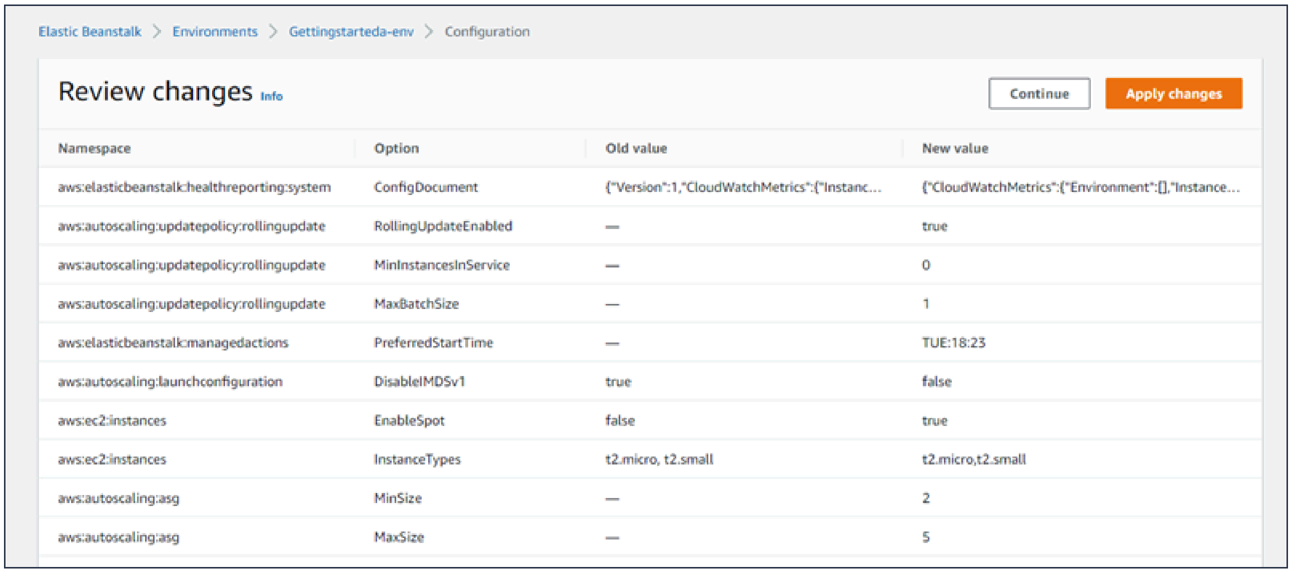
- **Cancel** – Go back to the environment's dashboard without applying your configuration changes. When you choose **Cancel**, the console loses any pending changes you made on any configuration category.

  You can also cancel your configuration changes by choosing another item on the left navigation page, like **Events** or **Logs**.
- **Review changes** – Get a summary of all the pending changes you made in any of the configuration categories. For details, see Review changes page.
- **Apply changes** – Apply the changes you made in any of the configuration categories to your environment. In some cases you're prompted to confirm a consequence of one of your configuration decisions.

Review changes page

The **Review Changes** page displays a table showing all the pending option changes you made in any of the configuration categories and haven't applied to your environment yet.

The tables lists each option as a combination of the **Namespace** and **Option** with which Elastic Beanstalk identifies it. For details, see Configuration options.


Elastic Beanstalk > Environments > Gettingstarteda-env > Configuration
Review changes Info
Continue
Apply changes

Namespace	Option	Old value	New value
aws:elasticbeanstalk:healthreporting:system	ConfigDocument	{"Version":1,"CloudWatchMetrics":{"Instanc...	{"CloudWatchMetrics":{"Environment":[],"Instance...
aws:autoscaling:updatepolicy:rollingupdate	RollingUpdateEnabled	—	true
aws:autoscaling:updatepolicy:rollingupdate	MinInstancesInService	—	0
aws:autoscaling:updatepolicy:rollingupdate	MaxBatchSize	—	1
aws:elasticbeanstalk:managedactions	PreferredStartTime	—	TUE:18:23
aws:autoscaling:launchconfiguration	DisableIMDSv1	true	false
aws:ec2:instances	EnableSpot	false	true
aws:ec2:instances	InstanceTypes	t2.micro, t2.small	t2.micro,t2.small
aws:autoscaling:asg	MinSize	—	2
aws:autoscaling:asg	MaxSize	—	5

When you're done reviewing your changes, you can choose one of the following actions:

- **Continue** – Go back to the **Configuration** overview page. You can then continue making changes or apply pending ones.
- **Apply changes** – Apply the changes you made in any of the configuration categories to your environment. In some cases you're prompted to confirm a consequence of one of your configuration decisions.

The Amazon EC2 instances for your Elastic Beanstalk environment

This topic explains the Amazon EC2 instances and the configuration options that affect your Elastic Beanstalk environment.

When you create a web server environment, AWS Elastic Beanstalk creates one or more Amazon Elastic Compute Cloud (Amazon EC2) virtual machines, known as *Instances*.

The instances in your environment are configured to run web apps on the platform that you choose. You can make changes to various properties and behaviors of your environment's instances when you create your environment or after it's already running. Or, you can already make these changes by modifying the source code that you deploy to the environment. For for more information, see the section called "Configuration options".

Note

The Auto Scaling group in your environment manages the Amazon EC2 instances that run your application. When you make configuration changes that are described in this topic, the launch configuration also changes. The launch configuration is either an Amazon EC2 launch template or an Auto Scaling group launch configuration resource. This change requires replacement of all instances. It also triggers either a rolling update or immutable update, depending on which one is configured.

EC2 instance purchasing options

Elastic Beanstalk supports several Amazon EC2 instance purchasing options:

- **On-Demand** — An *On-Demand Instance* is a pay-as-you-go resource—there's no long-term commitment required when you use it.
- **Reserved** — A *Reserved Instance* is a pre-purchased billing discount applied automatically to matching On-Demand instances in your environment.
- **Spot** — A *Spot Instance* is an unused Amazon EC2 instance that is available for less than the On-Demand price. You can enable and configure the allocation of Spot Instances in your environment. For more information, see Auto Scaling your Elastic Beanstalk environment instances.

Topics

Amazon EC2 instance types

This topic explains the term *instance type*. When you create a new environment, Elastic Beanstalk provisions Amazon EC2 instances that are based on the Amazon EC2 *instance types* that you choose. The instance types that you choose determine the host hardware that runs your instances. EC2 instance types can be categorized by which processor architecture each is based on. Elastic Beanstalk supports instance types based on the following processor architectures: AWS Graviton 64-bit Arm architecture (arm64), 64-bit architecture (x86), and 32-bit architecture (i386). Elastic Beanstalk selects the x86 processor architecture by default when you create a new environment.

Note

The i386 32-bit architecture is no longer supported by the majority of Elastic Beanstalk platforms. We recommended that you choose the x86 or arm64 architecture types instead. Elastic Beanstalk provides configuration options for i386 processor instance types in the `aws:ec2:instances` namespace.

All of the instance types in the configuration for a given Elastic Beanstalk environment must have the same type of processor architecture. Assume that you add a new instance type to an existing environment that already has a t2.medium instance type, which is based on x86 architecture. You can only add another instance type of the same architecture, such as t2.small. If you want to replace the existing instance types with those from a different architecture, you can do so. But make sure that all of the instance types in the command are based on the same type of architecture.

Elastic Beanstalk regularly adds support for new compatible instance types after Amazon EC2 introduces them. For information about instance types that are available, see Instance types in the *Amazon EC2 User Guide*.

Note

Elastic Beanstalk now offers support for Graviton on all of the latest Amazon Linux 2 platforms across all AWS Graviton supported Regions. For more information about creating an Elastic Beanstalk environment with arm64 based instances types, see Configuring Amazon EC2 instances using the Elastic Beanstalk console.
Create new environments that run Amazon EC2 instances on arm64 architecture and migrate your existing applications to them with the deployment options in Elastic Beanstalk.
To learn more about Graviton arm64 based processors, see these AWS resources:

- Benefits — The AWS Graviton Processor
- *Getting started* and other topics, such as *Language-specific considerations* — Getting started with AWS Graviton GitHub article

Configuring Amazon EC2 instances using the Elastic Beanstalk console

You can create or modify your Elastic Beanstalk environment's Amazon EC2 instance configuration in the Elastic Beanstalk console.

Note

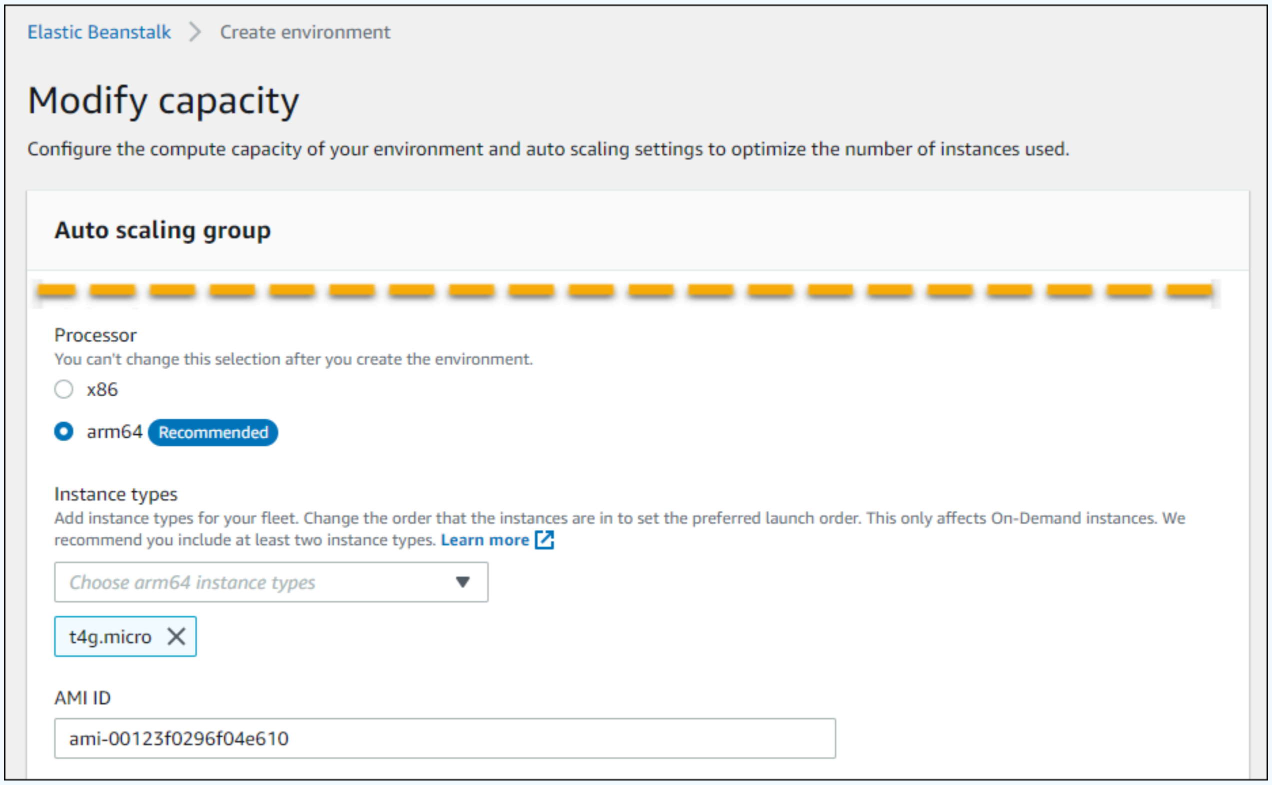
Although the Elastic Beanstalk console doesn't provide the option to change the processor architecture of an existing environment, you can do so with the AWS CLI. For example commands, see Configuring Amazon EC2 security groups and instance types using the AWS CLI.

To configure Amazon EC2 instances in the Elastic Beanstalk console during environment creation

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**.
3. Choose Create a new environment to start creating your environment.
4. On the wizard's main page, before choosing **Create environment**, choose **Configure more options**.
5. In the **Instances** configuration category, choose **Edit**. Make changes to settings in this category, and then choose **Apply**. For setting descriptions, see the section the section called “Instances category settings” on this page.
6. In the **Capacity** configuration category, choose **Edit**. Make changes to settings in this category, and then choose **Continue**. For setting descriptions, see the section the section called “Capacity category settings” on this page.

   Selecting processor architecture

   Scroll down to **Processor** to select a processor architecture for your EC2 instances. The console lists processor architectures that are supported by the platform that you chose earlier in the **Create environment** panel.
   If you don't see the processor architecture that you need, return to the configuration category list to select a platform that supports it. From the **Modify Capacity** panel, choose **Cancel**. Then, choose **Change platform version** to choose new platform settings. Next, in the **Capacity** configuration category choose **Edit** tot see the processor architecture choices again.



Elastic Beanstalk > Create environment

Modify capacity

Configure the compute capacity of your environment and auto scaling settings to optimize the number of instances used.

Auto scaling group

Processor

You can't change this selection after you create the environment.

x86 [✓ arm64] Recommended

Instance types

Add instance types for your fleet. Change the order that the instances are in to set the preferred launch order. This only affects On-Demand instances. We recommend you include at least two instance types. Learn more

Choose arm64 instance types

t4g.micro

AMI ID

ami-00123f0296f04e610

7. Choose **Save**, and then make any other configuration changes that your environment requires.
8. Choose **Create environment**.

To configure a running environment's Amazon EC2 instances in the Elastic Beanstalk console

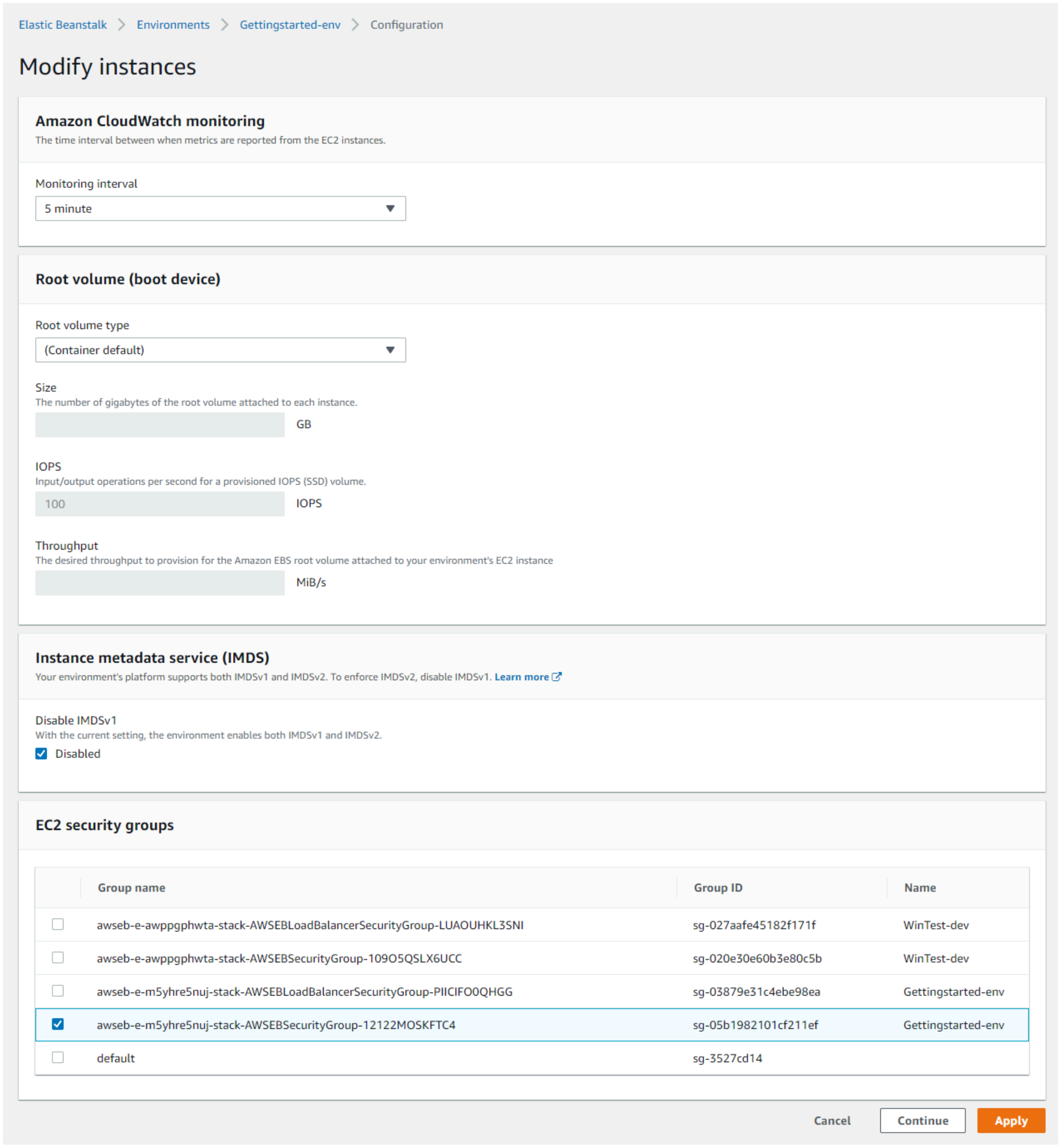
1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Instances** configuration category, choose **Edit**. Make changes to settings in this category, and then choose **Apply**. For setting descriptions, see the section the section called "Instances category settings" on this page.
5. In the **Capacity** configuration category, choose **Edit**. Make changes to settings in this category, and then choose **Continue**. For setting descriptions, see the section the section called "Capacity category settings" on this page.

Instances category settings

The following settings related to Amazon EC2 instances are available in the **Instances** configuration category.

Options

- Monitoring interval
- Root volume (boot device)
- Instance metadata service
- EC2 security groups


Elastic Beanstalk > Environments > Gettingstarted-env > Configuration

Modify instances

Amazon CloudWatch monitoring

The time interval between when metrics are reported from the EC2 instances.

Monitoring interval: 5 minute

Root volume (boot device)

Root volume type: (Container default)

Size

The number of gigabytes of the root volume attached to each instance.

GB

IOPS

Input/output operations per second for a provisioned IOPS (SSD) volume.

100 IOPS

Throughput

The desired throughput to provision for the Amazon EBS root volume attached to your environment's EC2 instance

MiB/s

Instance metadata service (IMDS)

Your environment's platform supports both IMDSv1 and IMDSv2. To enforce IMDSv2, disable IMDSv1. Learn more

Disable IMDSv1

With the current setting, the environment enables both IMDSv1 and IMDSv2.

[✓] Disabled

EC2 security groups

	Group name	Group ID	Name
☐	awseb-e-awppgphwta-stack-AWSEBLoadBalancerSecurityGroup-LUAOUHKL3SNI	sg-027aafe45182f171f	WinTest-dev
☐	awseb-e-awppgphwta-stack-AWSEBSecurityGroup-109O5QSLX6UCC	sg-020e30e60b3e80c5b	WinTest-dev
☐	awseb-e-m5yhre5nuj-stack-AWSEBLoadBalancerSecurityGroup-PIICIFO0QHGG	sg-03879e31c4ebe98ea	Gettingstarted-env
✓	awseb-e-m5yhre5nuj-stack-AWSEBSecurityGroup-12122MOSKFTC4	sg-05b1982101cf211ef	Gettingstarted-env
☐	default	sg-3527cd14	

Monitoring interval

By default, the instances in your environment publish basic health metrics to Amazon CloudWatch at five-minute intervals at no additional cost.

For more detailed reporting, you can set the **Monitoring interval** to **1 minute** to increase the frequency that the resources in your environment publish basic health metrics to CloudWatch at. CloudWatch service charges apply for one-minute interval metrics. For more information, see Amazon CloudWatch.

Root volume (boot device)

Each instance in your environment is configured with a root volume. The root volume is the Amazon EBS block device attached to the instance to store the operating system, libraries, scripts, and your application source code. By default, all platforms use general-purpose SSD block devices for storage.

You can modify **Root volume type** to use magnetic storage or provisioned IOPS SSD volume types and, if needed, increase the volume size. For provisioned IOPS volumes, you must also select the number of **IOPS** to provision. **Throughput** is only applicable to gp3 SSD volume types. You might enter the desired throughput to provision. It can range between 125 and 1000 mebibytes per second (MiB/s). Select the volume type that meets your performance and price requirements.

Important

The `RootVolumeType` option setting can cause Elastic Beanstalk to migrate an existing environment with launch configurations to launch templates. Doing so requires the necessary permissions to manage launch templates. These permissions are included in our managed policy. If you use custom policies instead of our managed policies, environment creation or updates might fail when you update your environment configuration. For more information and other considerations, see Migrating your Elastic Beanstalk environment to launch templates .

For more information, see Amazon EBS Volume Types in the *Amazon EC2 User Guide* and Amazon EBS Product Details.

Instance metadata service

The instance metadata service (IMDS) is an on-instance component that code on the instance uses to securely access instance metadata. Code can access instance metadata from a running instance using one of two methods. They are Instance Metadata Service Version 1 (IMDSv1) or Instance Metadata Service Version 2 (IMDSv2). IMDSv2 is more secure. Disable IMDSv1 to enforce IMDSv2. For more information, see the section called "IMDS".

Note

The IMDS section on this configuration page appears only for platform versions that support IMDSv2.

EC2 security groups

The security groups that are attached to your instances determine which traffic is allowed to reach and exit the instances.

The default EC2 security group that Elastic Beanstalk creates allows all incoming traffic from the internet or load balancers on the standard ports for HTTP (80) and SSH(22). You may also define your own custom security groups to designate firewall rules for the EC2 instances. The security groups can allow traffic on other ports or from other sources. For example, you can create a security group for SSH access that allows inbound traffic on port 22 from a restricted IP address range. Or for additional security, you can create one that allows traffic from a bastion host that only you can access.

You can select to opt out your environment from the default EC2 security group by setting the `DisableDefaultEC2SecurityGroup` option in the aws:autoscaling:launchconfiguration namespace to `true`. This option is not available in the console, but you can set it with the AWS CLI. For more information, see Managing EC2 security groups.

For more information about Amazon EC2 security groups, see Amazon EC2 Security Groups in the *Amazon EC2 User Guide*.

Note

To allow traffic between environment A's instances and environment B's instances, you can add a rule to the security group that Elastic Beanstalk attached to environment B. Then, you can specify the security group that Elastic Beanstalk attached to environment

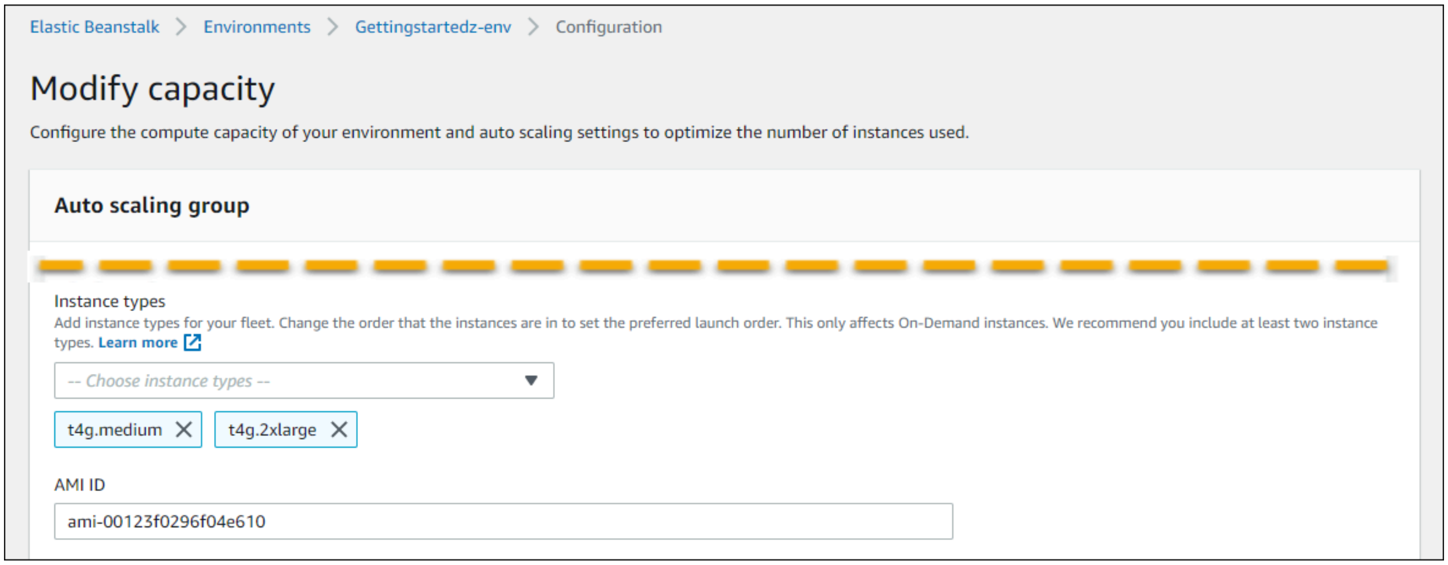
A. This allows inbound traffic from, or outbound traffic to, environment A's instances. However, doing so creates a dependency between the two security groups. If you later try to terminate environment A, Elastic Beanstalk can't delete the environment's security group, because environment B's security group is dependent on it.
Therefore, we recommend that you instead first create a separate security group. Then, attach it to environment A, and specify it in a rule of environment B's security group.

Capacity category settings

The following settings related to Amazon EC2 instances are available in the **Capacity** configuration category.

Options

- Instance types
- AMI ID


Elastic Beanstalk > Environments > Gettingstartedz-env > Configuration

Modify capacity

Configure the compute capacity of your environment and auto scaling settings to optimize the number of instances used.

Auto scaling group

Instance types

Add instance types for your fleet. Change the order that the instances are in to set the preferred launch order. This only affects On-Demand instances. We recommend you include at least two instance types. Learn more

-- Choose instance types --

t4g.medium

t4g.2xlarge

AMI ID

ami-00123f0296f04e610

Instance types

The **Instance types** setting determines the type of Amazon EC2 instance that's launched to run your application. This configuration page shows a list of **Instance types**. You can select one or more instance types. The Elastic Beanstalk console only displays the instance types based on the processor architecture that's configured for your environment. Therefore, you can only add instance types of the same processor architecture.

Note

Although the Elastic Beanstalk console doesn't provide the option to change the processor architecture of an existing environment, you can do so with the AWS CLI. For example commands, see Configuring Amazon EC2 security groups and instance types using the AWS CLI.

Choose an instance that's powerful enough to run your application under load, but not so powerful that it's idle most of the time. For development purposes, the t2 family of instances provides a moderate amount of power with the ability to burst for short periods of time. For large-scale, high-availability applications, use a pool of instances to ensure that capacity isn't too strongly affected if any single instance goes down. Start with an instance type that you can use to run five instances under moderate loads during normal hours. If any instance fails, the rest of the instances can absorb the rest of the traffic. The capacity buffer also allows time for the environment to scale up as traffic begins to rise during peak hours.

For more information about Amazon EC2 instance families and types, see Instance types in the *Amazon EC2 User Guide*. To determine which instance types meet your requirements and their supported Regions, see Available instance types in the *Amazon EC2 User Guide*.

AMI ID

The Amazon Machine Image (AMI) is the Amazon Linux or Windows Server machine image that Elastic Beanstalk uses to launch Amazon EC2 instances in your environment. Elastic Beanstalk provides machine images that contain the tools and resources required to run your application.

Elastic Beanstalk selects a default AMI for your environment based on the Region, platform version and processor architecture that you choose. If you have created a custom AMI, replace the default AMI ID with your own default custom one.

Managing EC2 security groups

When Elastic Beanstalk creates an environment, it assigns a default security group to the EC2 instances that are launched with it. The security groups that are attached to your instances determine which traffic is allowed to reach and exit the instances.

The default EC2 security group that Elastic Beanstalk creates allows all incoming traffic from the internet or load balancers on the standard ports for HTTP (80) and SSH(22). You may also define your own custom security groups to designate firewall rules for the EC2 instances. The security groups can allow traffic on other ports or from other sources. For example, you can create a security group for SSH access that allows inbound traffic on port 22 from a restricted IP address range. Or for additional security, you can create one that allows traffic from a bastion host that only you can access.

You can select to opt out your environment from the default EC2 security group by setting the `DisableDefaultEC2SecurityGroup` option in the aws:autoscaling:launchconfiguration namespace to `true`. Use the AWS CLI or configuration files to apply this option to your environment and to attach custom security groups to the EC2 instances.

Managing EC2 security groups in multi-instance environments

If you create a custom EC2 security group in a multi-instance environment you must also consider how the load balancers and incoming traffic rules keep your instances secure and accessible.

Inbound traffic to an environment with multiple EC2 instances is managed by the load balancer, which directs incoming traffic among all of the EC2 instances. When Elastic Beanstalk creates a default EC2 security group, it also defines inbound rules that allow incoming traffic from the load balancer. Without this inbound rule in the security group, the incoming traffic will not be allowed to enter the instances. This condition would essentially block the instances from external requests.

If you disable the default EC2 security group for a load balanced environment, Elastic Beanstalk validates some configuration rules. If the configuration doesn't meet the validation checks, it issues messages instructing you to provide the required configuration. The validation checks are the following:

- At least one security group must be assigned to the load balancer using the `SecurityGroups` option of the aws:elbv2:loadbalancer or aws:elb:loadbalancer, depending on whether it's an application load balancer or classic load balancer, respectively. For AWS CLI examples see Configuring with the AWS CLI.
- Inbound traffic rules must exist that allow your EC2 instances to receive traffic from the load balancer. Both your EC2 security groups and your load balancer security groups must reference these inbound rules. For more information, see the Inbound rules for traffic section that follows.

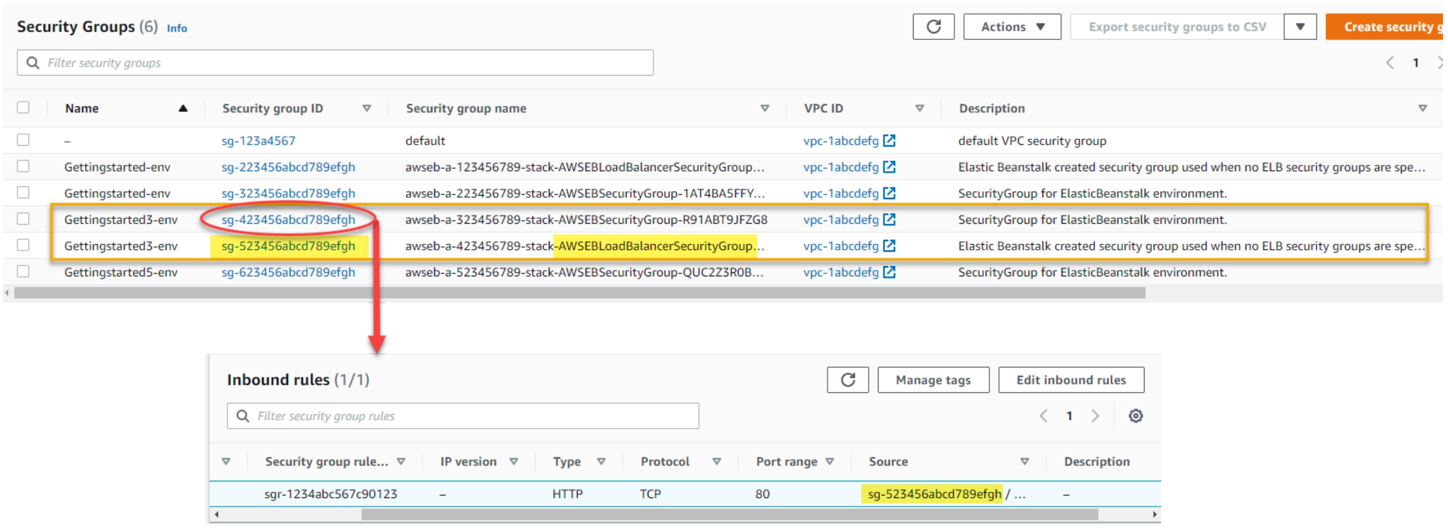
Inbound rules for traffic

The EC2 security group(s) for a multi-instance environment, must include an inbound rule that references the load balancer security group. This applies to environments with any type of load balancer, dedicated or shared, and with either custom or default load balancer security groups.

You can view all of the security groups that are attached to your environment components in the EC2 console. The following image shows the EC2 console listing of security groups that Elastic Beanstalk creates by default during the create environment operation.

The **Security Groups** screen shows environments and their associated security groups. Both *GettingStarted-env* and *GettingStarted3-env* are multi-instance environments with dedicated load balancers. Each of these environments has two security groups listed, one for the EC2 instances and another for the load balancer. Elastic Beanstalk creates these security groups when it creates the environments. *GettingStarted5-env* doesn't have a load balancer security group, because it only has one EC2 instance, and thus no load balancer.

The **Inbound rules** screen drills down into the EC2 security group for the instances of *GettingStarted3-env*. This example defines the inbound rules for the EC2 security group. Note that the *Source* column in the *Inbound rules* lists the security group id of the load balancer security group listed in the prior image. This rule allows the EC2 instances of *GettingStarted3-env* to receive inbound traffic from that specific load balancer on port 80.



Name	Security group ID	Security group name	VPC ID	Description
–	sg-123a4567	default	vpc-1abcdefg	default VPC security group
Gettingstarted-env	sg-223456abcd789efgh	awseb-a-123456789-stack-AWSEBLoadBalancerSecurityGroup…	vpc-1abcdefg	Elastic Beanstalk created security group used when no ELB security groups are spe…
Gettingstarted-env	sg-323456abcd789efgh	awseb-a-223456789-stack-AWSEBSecurityGroup-1AT4BA5FFY…	vpc-1abcdefg	SecurityGroup for ElasticBeanstalk environment.
Gettingstarted3-env	sg-423456abcd789efgh	awseb-a-323456789-stack-AWSEBSecurityGroup-R91ABT9JFZG8	vpc-1abcdefg	SecurityGroup for ElasticBeanstalk environment.
Gettingstarted3-env	sg-523456abcd789efgh	awseb-a-423456789-stack-AWSEBLoadBalancerSecurityGroup…	vpc-1abcdefg	Elastic Beanstalk created security group used when no ELB security groups are spe…
Gettingstarted5-env	sg-623456abcd789efgh	awseb-a-523456789-stack-AWSEBSecurityGroup-QUC2Z3R0B…	vpc-1abcdefg	SecurityGroup for ElasticBeanstalk environment.

Security group rule…	IP version	Type	Protocol	Port range	Source	Description
sgr-1234abc567c90123	–	HTTP	TCP	80	sg-523456abcd789efgh / …	–

For more information, see [Change security groups for your instance](#) and [Elastic Load Balancing rules](#) in the *Amazon EC2 User Guide*.

Configuring Amazon EC2 security groups and instance types using the AWS CLI

You can use the AWS Command Line Interface (AWS CLI) to configure the Amazon EC2 instances in your Elastic Beanstalk environments.

Configuring EC2 security groups using the AWS CLI

This topic provides examples for different EC2 security group configurations for both single-instance and load balanced (multi-instance) environments. For more information about the options in these examples, see aws:autoscaling:launchconfiguration.

Notes

The create environment operation provides an EC2 security group by default. It also creates an environment with an application load balancer by default.
The update environment operation can be used to either disable or enable the default EC2 security group for your environment with the boolean option `DisableDefaultEC2SecurityGroup`. *Example 5* shows how to set your environment back to the default security configuration if you had previously modified it.

The following examples show a create-environment command opting out of the default EC2 security group and providing custom security groups instead. Since the `DisableDefaultEC2SecurityGroup` option is set to `true`, the default EC2 security group that Elastic Beanstalk normally associates to the EC2 instances is not created. Therefore, you must provide other security groups with the `SecurityGroups` option.

Note that the aws:elasticbeanstalk:environment `EnvironmentType` option is set to `SingleInstance`. To create a single instance environment, you must specify this option, because `LoadBalanced` is the default `EnvironmentType`. Since this environment does not include a load balancer, we don't need to specify a load balancer security group.

Example 1 — New single-instance environment with custom EC2 security groups (namespace options inline)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v6.5.0 applrunning Node.js 22" \
--option-settings \
Namespace=aws:elasticbeanstalk:environment,OptionName=EnvironmentType,Value=SingleInstance
 \
Namespace=aws:autoscaling:launchconfiguration,OptionName=IamInstanceProfile,Value=aws-
elasticbeanstalk-ec2-role \
Namespace=aws:autoscaling:launchconfiguration,OptionName=DisableDefaultEC2SecurityGroup,Value=t
 \
Namespace=aws:autoscaling:launchconfiguration,OptionName=SecurityGroups,Value=sg-
abcdef01, sg-abcdef02 \
Namespace=aws:autoscaling:launchconfiguration,OptionName=EC2KeyName,Value=my-keypair
```

As an alternative, use an `options.json` file to specify the namespace options instead of including them inline.

Example 2 — New single-instance environment with custom EC2 security groups (namespace options in `options.json` file)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v6.5.0 running Node.js 22" \
--option-settings file://options.json
```

Example

```
### example options.json ###
[
  { "Namespace" : "aws:elasticbeanstalk:environment",
    "OptionName" : "EnvironmentType",
    "Value" : "SingleInstance"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "IamInstanceProfile",
    "Value": "aws-elasticbeanstalk-ec2-role"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "DisableDefaultEC2SecurityGroup",
    "Value": "true"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "SecurityGroups",
    "Value": "sg-abcdef01, sg-abcdef02"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "EC2KeyName",
    "Value": "my-keypair"
  }
]
```

The following example creates a load-balanced environment. It specifies the aws:elasticbeanstalk:environment namespace option `LoadBalancerType` set to `application`. Since we're disabling the default EC2 security group with the `DisableDefaultEC2SecurityGroup` option, we need to provide our own custom security groups for the EC2 instances again, with the aws:autoscaling:launchconfiguration `SecurityGroups` option, like the previous example. Since this environment has a load balancer to route traffic, we must provide security groups for the load balancer as well.

To create an environment with a with a classic load balancer, but otherwise the same configuration, update the configuration for the aws:elasticbeanstalk:environment namespace option `LoadBalancerType` to `classic`.

The different load balancer types have different namespaces that hold the options to specify the security groups:

- application load balancer – aws:elbv2:loadbalancer `SecurityGroups` option
- classic load balancer – aws:elb:loadbalancer `SecurityGroups` option
- network load balancer – since network load balancers do not have security groups, configure the EC2 security groups with VPC identifiers. For more information, see Update the security groups for your Network Load Balancer in the *User Guide for Network Load Balancers*.

Example 3 — New multi-instance environment with custom EC2 security groups (namespace options in `options.json` file)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v6.5.0 running Node.js 22" \
--option-settings file://options.json
```

Example

```
### example options.json ###
[
  {
    "Namespace" : "aws:elasticbeanstalk:environment",
    "OptionName" : "EnvironmentType",
    "Value" : "LoadBalanced"
  },
  {
  "Namespace" : "aws:elasticbeanstalk:environment",
    "OptionName" : "LoadBalancerType",
    "Value" : "application"
  },
  {
    "Namespace" : "aws:elbv2:loadbalancer",
    "OptionName" : "SecurityGroups",
    "Value" : "sg-abcdefghikl012345"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "IamInstanceProfile",
    "Value": "aws-elasticbeanstalk-ec2-role"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "DisableDefaultEC2SecurityGroup",
    "Value": "true"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "SecurityGroups",
    "Value": "sg-abcdef01, sg-abcdef02"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "EC2KeyName",
    "Value": "my-keypair"
  }
]
```

You can disable the default EC2 security group for an existing environment with the update-environment command. The following example command disables the default EC2 security group and assigns the environment's EC2 instances custom EC2 security groups.

Use the example `options.jason` files in examples 4(a), 4(b), or 4(c), depending on whether the environment is load balanced and the type of load balancer. Configuration file 4(a) specifies the security groups for a single-instance environment. Since it doesn't require a load balancer, we only provide the security group for the EC2 instances. Configuration files 4(b) and 4(c) specify the security groups for an application load balancer and a classic load balancer. For these cases we also need to specify security groups for the load balancer.

Example 4 — Update an existing environment to disable default EC2 security group (namespace options in `options.json` file)

```
aws elasticbeanstalk update-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v6.5.0 running Node.js 22" \
--option-settings file://options.json
```

Example 4(a) — Configuration file for single-instance environment (no load balancer)

```
### example options.json ###
[
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "DisableDefaultEC2SecurityGroup",
    "Value": "true"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "SecurityGroups",
    "Value": "sg-abcdef01, sg-abcdef02"
  }
]
```

To update an environment that uses an application load balancer, use the `aws:elbv2:loadbalancer` namespace to specify the security groups for the load balancer.

Example 4(b) — Configuration file for environment with an application load balancer

```
### example options.json ###
[
  {
    "Namespace" : "aws:elbv2:loadbalancer",
    "OptionName" : "SecurityGroups",
    "Value" : "sg-abcdefghikl012345"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "DisableDefaultEC2SecurityGroup",
    "Value": "true"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "SecurityGroups",
    "Value": "sg-abcdef01, sg-abcdef02"
  }
]
```

To update an environment that uses a classic load balancer use the `aws:elb:loadbalancer` namespace to specify the security groups for the load balancer.

Example 4(c) — Configuration file for environment with a classic load balancer

```
### example options.json ###
[
  {
    "Namespace" : "aws:elb:loadbalancer",
    "OptionName" : "SecurityGroups",
    "Value" : "sg-abcdefghikl012345"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "DisableDefaultEC2SecurityGroup",
    "Value": "true"
  },
  {
    "Namespace": "aws:autoscaling:launchconfiguration",n
    "OptionName": "SecurityGroups",
    "Value": "sg-abcdef01, sg-abcdef02"
  }
]
```

To return your environment to the default behavior and configuration with the default security group that Elastic Beanstalk assigns, use the update-environment command to set the `DisableDefaultEC2SecurityGroup` to `false`. For a multi-instance environment, Elastic Beanstalk also handles the security groups and network traffic rules for your environment's load balancer.

The following example applies to both a single-instance or multi-instance (load balanced) environment:

Example 5 — Update an environment back to using the default security group (namespace options in `options.json` file)

```
aws elasticbeanstalk update-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v6.5.0 running Node.js 22" \
--option-settings file://options.json
```

Example

```
### example options.json ###
[
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "DisableDefaultEC2SecurityGroup",
    "Value": "false"
  }
]
```

Configuring EC2 with instance types using the AWS CLI

This topic provides examples for configuring the instance types of the EC2 instances in your environment.

The first two examples creates a new environment. The command specifies an Amazon EC2 instances type, t4g.small, that's based on arm64 processor architecture. Elastic Beanstalk defaults the Image ID (AMI) for the EC2 instances based on the Region, platform version and instance type.

The instance type corresponds to a processor architecture. The `solution-stack-name` parameter applies to platform version.

Example 1 — create a new arm64 based environment (namespace options inline)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2 v3.4.7 running Docker" \
--option-settings \
Namespace=aws:autoscaling:launchconfiguration,OptionName=IamInstanceProfile,Value=aws-
elasticbeanstalk-ec2-role \
Namespace=aws:ec2:instances,OptionName=InstanceTypes,Value=t4g.small
```

As an alternative, use an `options.json` file to specify the namespace options instead of including them inline.

Example 2 — create a new arm64 based environment (namespace options in `options.json` file)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2 v3.4.7 running Docker" \
--option-settings file://options.json
```

Example

```
### example options.json ###
[
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "IamInstanceProfile",
    "Value": "aws-elasticbeanstalk-ec2-role"
  },
  {
    "Namespace": "aws:ec2:instances",
    "OptionName": "InstanceTypes",
    "Value": "t4g.small"
  }
]
```

The next two examples update the configuration for an existing environment with the update-environment command. In this example we're adding another instance type that's also based on arm64 processor architecture. For existing environments, all instance types that are added must have the same processor architecture. If you want to replace the existing instance types with those from a different architecture, you can do so. But make sure that all of the instance types in the command have the same type of architecture.

Example 3 — update an existing arm64 based environment (namespace options inline)

```
aws elasticbeanstalk update-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2 v3.4.7 running Docker" \
--option-settings \
Namespace=aws:autoscaling:launchconfiguration,OptionName=IamInstanceProfile,Value=aws-elasticbeanstalk-ec2-role \
Namespace=aws:ec2:instances,OptionName=InstanceTypes,Value=t4g.small,t4g.micro
```

As an alternative, use an `options.json` file to specify the namespace options instead of including them inline.

Example 4 — update an existing arm64 based environment (namespace options in `options.json` file)

```
aws elasticbeanstalk update-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2 v3.4.7 running Docker" \
--option-settings file://options.json
```

Example

```
### example options.json ###
```

```
[
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "IamInstanceProfile",
    "Value": "aws-elasticbeanstalk-ec2-role"
  },
  {
    "Namespace": "aws:ec2:instances",
    "OptionName": "InstanceTypes",
    "Value": "t4g.small, t4g.micro"
  }
]
```

The next two examples show more create-environment commands. These examples don't provide values for `InstanceTypes`. When `InstanceTypes` values aren't specified, Elastic Beanstalk defaults to x86 based processor architecture. The Image ID (AMI) for the environment's EC2 instances will default according to the Region, platform version and defaulted instance type. The instance type corresponds to a processor architecture.

Example 5 — create a new x86 based environment (namespace options inline)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2 v3.4.7 running Docker" \
--option-settings \
Namespace=aws:autoscaling:launchconfiguration,OptionName=IamInstanceProfile,Value=aws-
elasticbeanstalk-ec2-role
```

As an alternative, use an `options.json` file to specify the namespace options instead of including them inline.

Example 6 — create a new x86 based environment (namespace options in `options.json` file)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2 v3.4.7 running Docker" \
--option-settings file://options.json
```

Example

```
### example options.json ###
[
  {
    "Namespace": "aws:autoscaling:launchconfiguration",
    "OptionName": "IamInstanceProfile",
    "Value": "aws-elasticbeanstalk-ec2-role"
  }
]
```

Configuring Amazon EC2 instances with namespace options

You can use the configuration options in the `aws:autoscaling:launchconfiguration` namespace to configure the instances for your environment, including additional options that aren't available in the console.

Important

The `DisableIMDSv1`, `RootVolumeType`, or `BlockDeviceMappings` option setting can cause Elastic Beanstalk to migrate an existing environment with launch configurations to launch templates. Doing so requires the necessary permissions to manage launch templates. These permissions are included in our managed policy. If you use custom policies instead of our managed policies, environment creation or updates might fail when you update your environment configuration. For more information and other considerations, see Migrating your Elastic Beanstalk environment to launch templates .

The following configuration file example uses the basic configuration options that are explained in this topic. To see examples of additional configuration options when you need to specify security groups for load balancers, see Configuring with the AWS CLI.

```
option_settings:
  aws:autoscaling:launchconfiguration:
    SecurityGroups: my-securitygroup
    MonitoringInterval: "1 minute"
    DisableIMDSv1: false
    DisableDefaultEC2SecurityGroup: true
    SecurityGroups: "sg-abcdef01, sg-abcdef02"
    EC2KeyName: my-keypair
    IamInstanceProfile: "aws-elasticbeanstalk-ec2-role"
    BlockDeviceMappings: "/dev/sdj=:100,/dev/sdh=snap-51eef269,/dev/sdb=ephemeral0"
  aws:elasticbeanstalk:environment:
    EnvironmentType: SingleInstance
```

The `DisableDefaultEC2SecurityGroup` and `BlockDeviceMappings` are not available in the console.

You can use `BlockDeviceMappings` to configure additional block devices for your instances. For more information, see Block Device Mapping in the *Amazon EC2 User Guide*.

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See Recommended values for details.

Configuring the IMDS on your Elastic Beanstalk environment's instances

This topic describes the Instance Metadata Service (IMDS).

Instance metadata is data that's related to an Amazon Elastic Compute Cloud (Amazon EC2) instance that applications can use to configure or manage the running instance. The instance metadata service (IMDS) is an on-instance component that code on the instance uses to securely access instance metadata. This code can be Elastic Beanstalk platform code on your environment instances, the AWS SDK that your application might be using, or even your application's own code. For more information, see Instance metadata and user data in the *Amazon EC2 User Guide*.

Code can access instance metadata from a running instance using one of two methods: Instance Metadata Service Version 1 (IMDSv1) or Instance Metadata Service Version 2 (IMDSv2). IMDSv2 uses session-oriented requests and mitigates several types of vulnerabilities that could be used to try to access the IMDS. For information about these two methods, see Configuring the instance metadata service in the *Amazon EC2 User Guide*.

Topics

- The aws:autoscaling:launchconfiguration namespace

Platform support for IMDS

Elastic Beanstalk platforms running on Amazon Linux 2 and Amazon Linux 2023 and Windows server all support both IMDSv1 and IMDSv2. For more information, see Configuring IMDS using the Elastic Beanstalk console

Choosing IMDS methods

When making a decision about the IMDS methods that you want your environment to support, consider the following use cases:

- *AWS SDK* – If your application uses an AWS SDK, make sure you use an the latest version of the SDK. The AWS SDKs make IMDS calls, and newer SDK versions use IMDSv2 whenever possible. If you ever disable IMDSv1, or if your application uses an old SDK version, IMDS calls might fail.
- *Your application code* – If your application makes IMDS calls, consider using the AWS SDK so that you can make the calls instead of making direct HTTP requests. This way, you don't need to make code changes to switch between IMDS methods. The AWS SDK uses IMDSv2 whenever possible.
- *Elastic Beanstalk platform code* – Our code makes IMDS calls through the AWS SDK, and therefore uses IMDSv2 on all supporting platform versions. If your code uses an up-to-date AWS SDK and makes all IMDS calls through the SDK, you can safely disable IMDSv1.

Configuring IMDS using the Elastic Beanstalk console

You can modify your Elastic Beanstalk environment's Amazon EC2 instance configuration in the Elastic Beanstalk console.

Important

The `DisableIMDSv1` option setting can cause Elastic Beanstalk to migrate an existing environment with launch configurations to launch templates. Doing so requires the necessary permissions to manage launch templates. These permissions are included in our managed policy. If you use custom policies instead of our managed policies, environment creation or updates might fail when you update your environment configuration. For more information and other considerations, see Migrating your Elastic Beanstalk environment to launch templates .

To configure IMDS on your Amazon EC2 instances in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Instance traffic and scaling** configuration category, choose **Edit**.
5. Set **Disable IMDSv1** to enforce IMDSv2. Clear **Disable IMDSv1** to enable both IMDSv1 and IMDSv2.
6. To save the changes choose **Apply** at the bottom of the page.

The aws:autoscaling:launchconfiguration namespace

You can use a configuration option in the `aws:autoscaling:launchconfiguration` namespace to configure IMDS on your environment's instances.

Important

The `DisableIMDSv1` option setting can cause Elastic Beanstalk to migrate an existing environment with launch configurations to launch templates. Doing so requires the necessary permissions to manage launch templates. These permissions are included in our managed policy. If you use custom policies instead of our managed policies, environment creation or updates might fail when you update your environment configuration. For more information and other considerations, see Migrating your Elastic Beanstalk environment to launch templates .

The following configuration file example disables IMDSv1 using the `DisableIMDSv1` option.

```
option_settings:
  aws:autoscaling:launchconfiguration:
    DisableIMDSv1: true
```

Set **DisableIMDSv1** to `true` to disable IMDSv1 and enforce IMDSv2.

Set **DisableIMDSv1** to `false` to enable both IMDSv1 and IMDSv2.

Auto Scaling your Elastic Beanstalk environment instances

This topic describes how you can customize the Auto Scaling features to manage your Elastic Beanstalk environment's workload. You can configure Auto Scaling for your environment using the Elastic Beanstalk console, namespace configuration options, the AWS CLI, or the EB CLI.

Load-balanced or single instance environments

Your AWS Elastic Beanstalk environment includes an *Auto Scaling group* that manages the Amazon EC2 instances in your environment. In a single-instance environment, the Auto Scaling group ensures that there is always one instance running. In a load-balanced environment, you configure the group with a range of instances to run, and Auto Scaling adds or removes instances as needed, based on load.

EC2 Instance configuration

The Auto Scaling group also applies your configuration choices to provision and manage the EC2 instances in your environment. You can modify the EC2 configuration to change the instance type, key pair, Amazon Elastic Block Store (Amazon EBS) storage, and other settings that can only be configured when you launch an instance.

On-Demand and Spot Instances

As an option, Elastic Beanstalk can include Spot Instances in your environment and manage them in combination with On-Demand instances. You can configure Amazon EC2 Auto Scaling to monitor and automatically respond to changes that affect the availability of your Spot Instances by enabling Capacity Rebalancing. You can also configure the Spot allocation strategy that the Auto Scaling service uses to provision Spot Instances to your environment.

Required permissions when enabling Spot Instances

Enabling Spot Instance requests requires using Amazon EC2 launch templates. When you configure this feature during environment creation or updates, Elastic Beanstalk attempts to configure your environment to use Amazon EC2 launch templates (if the environment isn't using them already). In this case, if your user policy lacks the necessary permissions, environment creation or updates might fail. Therefore, we recommend that you use our managed user policy or add the required permissions to your custom policies. For details about the required permissions, see Required permissions for launch templates.

Auto Scaling triggers

The Auto Scaling group uses two Amazon CloudWatch alarms to trigger scaling operations. The default triggers scale when the average outbound network traffic from each instance is higher than 6 MiB or lower than 2 MiB over a period of five minutes. To use Auto Scaling effectively, configure triggers that are appropriate for your application, instance type, and service requirements. You can scale based on several statistics including latency, disk I/O, CPU utilization, and request count.

Schedule Auto Scaling actions

To optimize your environment's use of Amazon EC2 instances through predictable periods of peak traffic, configure your Auto Scaling group to change its instance count on a schedule. You can schedule changes to your group's configuration that recur daily or weekly, or schedule one-time changes to prepare for marketing events that will drive a lot of traffic to your site.

Auto Scaling health check

Auto Scaling monitors the health of each Amazon EC2 instance that it launches. If any instance terminates unexpectedly, Auto Scaling detects the termination and launches a replacement instance. To configure the group to use the load balancer's health check mechanism, see Auto Scaling health check setting for your Elastic Beanstalk environment.

Topics

Migrating your Elastic Beanstalk environment to launch templates

As of October 1, 2024, Amazon EC2 Auto Scaling no longer supports launch configurations for new accounts. Accounts created prior to that date might have launch configurations.

We recommend migrating to **launch templates** for the following benefits:

- Improved availability for your applications
- Better optimization of workloads in your Auto Scaling groups
- Access to the latest EC2 and Auto Scaling features

For more information, see Auto Scaling launch configurations in the *Amazon EC2 Auto Scaling User Guide*.

Option settings for launch templates

To migrate your environment from launch configurations to launch templates, set one of the following configuration options:

- `RootVolumeType` option set to **gp3**. You can set this option with the console or the namespace .
- `BlockDeviceMappings` option contains **gp3**. You can set this option with the console or the namespace.
- `DisableIMDSv1` option set to **true**. We recommend that you set this option using the namespace.
- `EnableSpot` option set to **true**. For more information, see Enabling Spot Instances.

Important

After an environment begins using launch templates, Elastic Beanstalk does not revert to launch configurations, even if you remove the configuration options that originally triggered the use of launch templates.

Confirm whether your environment has launch configurations or launch templates

You can confirm if your environment already uses launch templates, or if it's using launch configurations, by inspecting the CloudFormation stack template.

To inspect your environment's CloudFormation stack template

1. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. On the navigation bar at the top of the screen, choose the AWS Region where you created the environment.
3. On the **Stacks** page of the CloudFormation console, inspect the **Description** column.

   Locate and select the stack for the Elastic Beanstalk environment. CloudFormation displays the stack details for the environment.

4. In **Stack details** select the **Template** tab.

   Using your browser's page search, you can search the template text for *launchtemplate* or *launchconfiguration*.

For more information, see View stack information in the *AWS CloudFormation User Guide*.

Required permissions for launch templates

The default Elastic Beanstalk managed service role policy AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy provides the required permissions to create and manage launch templates. Elastic Beanstalk must manage launch templates to complete many environment operations, including creating environments.

If you attach custom policies to an Elastic Beanstalk service role, verify that the service role includes the following permissions for creating launch templates. These permissions enable Elastic Beanstalk to successfully create and update environments in your account:

Required permissions for Amazon EC2 launch templates

- `ec2:RunInstances`
- `ec2:CreateLaunchTemplate`
- `ec2:CreateLaunchTemplateVersions`
- `ec2:DeleteLaunchTemplate`
- `ec2:DeleteLaunchTemplateVersions`
- `ec2:DescribeLaunchTemplate`
- `ec2:DescribeLaunchTemplateVersions`

The following example IAM policy includes these permissions.

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:RunInstances",
        "ec2:CreateLaunchTemplate",
        "ec2:CreateLaunchTemplateVersions",
        "ec2:DeleteLaunchTemplate",
        "ec2:DeleteLaunchTemplateVersions",
        "ec2:DescribeLaunchTemplate",
        "ec2:DescribeLaunchTemplateVersions"
      ],
      "Resource": [
                "*"
            ]
     }
  ]
}
```

For more information, see Managing Elastic Beanstalk service roles and Managing Elastic Beanstalk user policies.

More about launch templates

To learn more about launch templates, see Auto Scaling launch templates in the *Amazon EC2 Auto Scaling User Guide*.

To learn more about the AWS transition to launch templates and the benefits they offer, see Amazon EC2 Auto Scaling will no longer add support for new EC2 features to Launch Configurations in the *AWS Compute Blog*.

Important

You don't need to follow the procedure referenced in this blog article to transition an older environment to launch templates. To migrate an existing Elastic Beanstalk environment to launch templates, set one of the options listed in Option settings for launch templates.

Spot Instance support for your Elastic Beanstalk environment

This topic describes the configuration options that are available for you to manage the capacity and load balancing of Spot Instances in your Elastic Beanstalk environment. It also provides details and examples for the methods you can use to configure these options. You can use the Elastic Beanstalk console, namespace configuration options, the AWS CLI, or the EB CLI to manage the configuration options.

Minimize Spot instance interruptions with Capacity Rebalancing

To help minimize the impact of Spot Instance interruptions to your application, you can enable the Capacity Rebalancing option included with Amazon EC2 Auto Scaling.

Important

Demand for Spot Instances can vary significantly from moment to moment, and the availability of Spot Instances can also vary significantly depending on how many unused Amazon EC2 instances are available. It's always possible that your Spot Instance might be interrupted.

When you enable Capacity Rebalancing, EC2 automatically attempts to replace Spot Instances in an Auto Scaling group before they are interrupted. To enable this feature use the Elastic Beanstalk console to configure the Auto Scaling group. Alternatively, you can set the Elastic Beanstalk `EnableCapacityRebalancing` configuration option to `true` in the aws:autoscaling:asg namespace.

For more information, see Capacity Rebalancing in the *Amazon EC2 Auto Scaling User Guide* and Spot Instance Interruptions in the *Amazon EC2 User Guide*.

Older Instance Types and Spot Instance Support

Some older AWS accounts might provide Elastic Beanstalk with default instance types that don't support Spot Instances. If you enable Spot Instance requests and you see the error None of the instance types you specified supports Spot, update your configuration with instance types that support Spot Instances. To choose Spot Instance types, use the Spot Instance Advisor.

Topics

Enabling Spot Instances for your environment

To take advantage of Amazon EC2 Spot Instances, set the `EnableSpot` option for your environment. Your environment's Auto Scaling group then combines Amazon EC2 purchase options and maintains a mix of On-Demand and Spot Instances.

You can use the Elastic Beanstalk console, namespace configuration options, the AWS CLI, or the EB CLI to enable Spot Instance requests for your environment.

Before you enable Spot Instances for your environment, become familiar with the Auto Scaling, capacity, and load balancing configuration options that are available. Your application's requirements that are related to workload, impact of instance interruptions, and pricing, are all important considerations in your planning to enable Spot Instances.

The topics that follow provide details about the Auto Scaling and capacity management options and how their combined use affects your environment. There are procedures and example configurations to inform and guide you about the various options and how to configure them. We also offer tools and features to help you manage your configuration and respond to events. You can schedule automated changes to your configuration based on predictable periods of traffic, configure triggers to respond to factors such as traffic volume, and configure Auto Scaling monitoring and health checks.

For more detailed information about Spot Instances, including explanation of key concepts and best practices, see Spot Instances in the *Amazon EC2 User Guide*.

Important

The `EnableSpot` option setting can cause Elastic Beanstalk to migrate an existing environment with launch configurations to launch templates. Doing so requires the necessary permissions to manage launch templates. These permissions are included in our managed policy. If you use custom policies instead of our managed policies, environment creation or updates might fail when you update your environment configuration. For more information and other considerations, see Migrating your Elastic Beanstalk environment to launch templates .

Spot Instance allocation strategy

You can select any one of the allocation strategies listed in this topic for your Elastic Beanstalk environment. Use the Elastic Beanstalk console, namespace configuration options, or the AWS CLI, to set and configure Spot Instance allocation strategy and related attributes for your environment.

Amazon EC2 applies an *allocation strategy* to manage and provision Spot instances for your environment. Each allocation strategy optimizes the allocated instances based on how it’s defined to handle available capacity, price, and selection of instance types.

Amazon EC2 Auto Scaling provides the following allocation strategies for Spot Instances.

- **Capacity optimized** (default)
  - Requests Spot Instances from the pool, with *optimal capacity* for the number of instances that are launching.
  - This strategy works well for workloads where the possibility of service disruption must be minimized.
- **Price capacity optimized**
  - Requests Spot Instances from the pools that have the *lowest chance of interruption* and the *lowest possible price*.
  - This is the preferable choice for most Spot workloads.
- **Capacity optimized prioritized**
  - Requests Spot Instances based on *capacity availability first*, while honoring your choice of *instance type prioritization* on a best-effort basis. You can provide a list of instance types, ordered by priority, when you configure Spot Instance options for Elastic Beanstalk.
  - This strategy is good for workloads that require minimal service disruption, and a specific instance type prioritization matters.
- **Lowest price**
  - Requests Spot Instances from the *lowest priced pool* with available instances.
  - It's important to take precaution when using this strategy, since it only considers instance price and not capacity availability, which will result in high interruption rates.

For more details about each allocation strategy, see Allocation strategies for multiple instance types in the *Amazon EC2 Auto Scaling User Guide*.

To help you understand which allocation strategy is best suited to meet your environment's requirements, see Choose the appropriate Spot allocation strategy in the *Amazon EC2 User Guide*.

Managing On-Demand instances and Spot instances

You can launch and automatically scale a fleet of On-Demand Instances and Spot Instances within a single Auto Scaling group. The following options can be used in tandem to configure how the Auto Scaling service manages Spot Instances and On-Demand Instances in your environment.

You can configure these options for your environment using the Elastic Beanstalk console, namespace configuration options, the AWS CLI, or the EB CLI.

These options are part of the aws:ec2:instances namespace:

- `EnableSpot` - When set to `true` this setting enables Spot Instance requests for your environment.
- `SpotFleetOnDemandBase` - Sets the minimum number of On-Demand Instances that your Auto Scaling group provisions before considering Spot Instances as your environment scales up.
- `SpotFleetOnDemandAboveBasePercentage` - The percentage of On-Demand Instances as part of additional capacity that your Auto Scaling group provisions beyond the `SpotOnDemandBase` instances.

The previously listed options correlate with the following options in the aws:autoscaling:asg namespace:

- `MinSize` - The minimum number of instances that you want in your Auto Scaling group.
- `MaxSize` - The maximum number of instances that you want in your Auto Scaling group.

Important

The `EnableSpot` option setting can cause Elastic Beanstalk to migrate an existing environment with launch configurations to launch templates. Doing so requires the necessary permissions to manage launch templates. These permissions are included in our managed policy. If you use custom policies instead of our managed policies, environment creation or updates might fail when you update your environment configuration. For more information and other considerations, see Migrating your Elastic Beanstalk environment to launch templates .

Applying both sets of namespace options

The following points describe how the combination of these option settings affects the scaling for your environment.

- Only `MinSize` determines your environment's initial capacity—the number of instances you want running at a minimum.

- `SpotFleetOnDemandBase` doesn't affect initial capacity. When Spot is enabled, this option determines how many On-Demand Instances are provisioned before any Spot Instances are considered.
- Consider when `SpotFleetOnDemandBase` is less than `MinSize`. You'll still get exactly `MinSize` instances as initial capacity. At least `SpotFleetOnDemandBase` of them must be On-Demand Instances.
- Consider when `SpotFleetOnDemandBase` is greater than `MinSize`. As your environment scales out, you're guaranteed to get at least an additional amount of instances equal to the difference between the two values. In other words, you're guaranteed to get at least an additional `(SpotFleetOnDemandBase - MinSize)` instances that are On-Demand before satisfying the `SpotFleetOnDemandBase` requirement.

Single-instance environments

In production environments, Spot Instances are particularly useful as part of a scalable, load-balanced environment. We don't recommend using Spot in a single-instance environment. If Spot Instances aren't available, you might lose the entire capacity (a single instance) of your environment. You may still wish to use a Spot Instance in a single instance environment for development or testing. When you do, be sure to set both `SpotFleetOnDemandBase` and `SpotFleetOnDemandAboveBasePercentage` to zero. Any other settings result in an On-Demand Instance.

Examples of scaling options settings

The following examples demonstrate different scenarios of setting the various scaling options. All examples assume a load-balanced environment with Spot Instance requests enabled.

Example 1: On-Demand and Spot as part of initial capacity

Option settings

Option	Namespace	Value
`MinSize`	`aws:autoscaling:asg`	`10`
`MaxSize`	`aws:autoscaling:asg`	`24`

Option	Namespace	Value
`SpotFleetOnDemandBase`	`aws:ec2:instances`	`4`
`SpotFleetOnDemandAboveBasePercentage`	`aws:ec2:instances`	`50`

In this example, the environment starts with ten instances, of which seven are On-Demand (four base, and 50% of the six above base) and three are Spot. The environment can scale out up to 24 instances. As it scales out, the portion of On-Demand in the part of the fleet above the four base On-Demand instances is kept at 50%, up to a maximum of 24 instances overall, of which 14 are On-Demand (four base, and 50% of the 20 above base) and ten are Spot.

Example 2: All On-Demand initial capacity

Option settings

Option	Namespace	Value
`MinSize`	`aws:autoscaling:asg`	`4`
`MaxSize`	`aws:autoscaling:asg`	`24`
`SpotFleetOnDemandBase`	`aws:ec2:instances`	`4`
`SpotFleetOnDemandAboveBasePercentage`	`aws:ec2:instances`	`50`

In this example, the environment starts with four instances, all of which are On-Demand. The environment can scale out up to 24 instances. As it scales out, the portion of On-Demand in the part of the fleet above the four base On-Demand instances is kept at 50%, up to a maximum of 24 instances overall, of which 14 are On-Demand (four base, and 50% of the 20 above base) and ten are Spot.

Example 3: Additional On-Demand base beyond initial capacity

Option settings

Option	Namespace	Value
`MinSize`	`aws:autoscaling:asg`	3
`MaxSize`	`aws:autoscaling:asg`	24
`SpotFleetOnDemandBase`	`aws:ec2:instances`	4
`SpotFleetOnDemandAboveBasePercentage`	`aws:ec2:instances`	50

In this example, the environment starts with three instances, all of which are On-Demand. The environment can scale out up to 24 instances. The first additional instance above the initial three is On-Demand, to complete the four base On-Demand instances. As it scales out further, the portion of On-Demand in the part of the fleet above the four base On-Demand instances is kept at 50%, up to a maximum of 24 instances overall, of which 14 are On-Demand (four base, and 50% of the 20 above base) and ten are Spot.

Capacity configuration for your Elastic Beanstalk environment

This topic describes the different approaches to configure Auto Scaling capacity for your Elastic Beanstalk environment. You can use the Elastic Beanstalk console, the EB CLI, the AWS CLI, or namespace options.

Important

The `EnableSpot` option setting can cause Elastic Beanstalk to migrate an existing environment with launch configurations to launch templates. Doing so requires the necessary permissions to manage launch templates. These permissions are included in our managed policy. If you use custom policies instead of our managed policies, environment creation or updates might fail when you update your environment configuration. For more information and other considerations, see Migrating your Elastic Beanstalk environment to launch templates .

Configuration using the console

You can configure the capacity management of an Auto Scaling group by editing **Capacity** on the environment's **Configuration** page in the Elastic Beanstalk console.

To configure Auto Scaling group capacity in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. In the **Auto Scaling group** section, configure the following settings.

   - **Environment type** – Select **Load balanced**.
   - **Min instances** – The minimum number of EC2 instances that the group should contain at any time. The group starts with the minimum count and adds instances when the scale-up trigger condition is met.
   - **Max instances** – The maximum number of EC2 instances that the group should contain at any time.

     Note

     If you use rolling updates, be sure that the maximum instance count is higher than the **Minimum instances in service** setting for rolling updates.

   - **Fleet composition** – The default is **On-Demand Instances**. To enable *Spot Instance* requests, select **Combined purchase options and instances**.

     Important

     The `EnableSpot` option setting can cause Elastic Beanstalk to migrate an existing environment with launch configurations to launch templates. Doing so requires the necessary permissions to manage launch templates. These permissions are included in our managed policy. If you use custom policies instead of our managed policies, environment creation or updates might fail when you update your environment configuration. For more information and other considerations, see Migrating your Elastic Beanstalk environment to launch templates .

The following options are enabled if you select to enable *Spot Instance* requests:

- **Spot allocation strategy** – Determines the method used to manage and provision the Spot Instances in your environment, based on available capacity, price, and selection of instance types. Select from *Capacity optimized* (default), *Price capacity optimized*, *Capacity optimized prioritized*, or *Lowest price*. For a description of each allocation strategy and more information, see the section called "Spot allocation strategy".
- **Maximum spot price** – For recommendations about maximum price options for Spot Instances, see Spot Instance pricing history in the *Amazon EC2 User Guide*.
- **On-Demand base** – The minimum number of On-Demand Instances that your Auto Scaling group provisions before considering Spot Instances as your environment scales out.
- **On-Demand above base** – The percentage of On-Demand Instances as part of any additional capacity that your Auto Scaling group provisions beyond the On-Demand base instances.

  Note

  The options **On-Demand base** and **On-Demand above base** correlate to the **Min** and **Max** *Instances* options listed earlier. For more information about these options and examples, see the section called "Spot Instance support".

- **Capacity Rebalancing** – This option is only relevant when there is at least one Spot Instance in your Auto Scaling group. When this feature is enabled, EC2 automatically attempts to replace Spot Instances in the Auto Scaling group before they're interrupted, minimizing Spot Instance interruptions to your applications. For more information, see Capacity Rebalancing in the *Amazon EC2 Auto Scaling User Guide*

- **Architecture** – The processor architecture for your EC2 instances. The processor architecture determines the EC2 Instance types that become available in the next field.
- **Instance types** – The types of Amazon EC2 instance launched to run your application. For details, see the section called "Instance types".

- **AMI ID** – The machine image that Elastic Beanstalk uses to launch Amazon EC2 instances in your environment. For details, see the section called "AMI ID".
- **Availability Zones** – Choose the number of Availability Zones to spread your environment's instances across. By default, the Auto Scaling group launches instances evenly across all usable zones. To concentrate your instances in fewer zones, choose the number of zones to use. For production environments, use at least two zones to ensure that your application is available in case one Availability Zone goes out.
- **Placement** (optional) – Choose the Availability Zones to use. Use this setting if your instances need to connect to resources in specific zones, or if you have purchased reserved instances, which are zone-specific. If you launch your environment in a custom VPC, you cannot configure this option. In a custom VPC, you choose Availability Zones for the subnets that you assign to your environment.
- **Scaling cooldown** – The amount of time, in seconds, to wait for instances to launch or terminate after scaling, before continuing to evaluate triggers. For more information, see Scaling Cooldowns.

6. To save the changes choose **Apply** at the bottom of the page.

Configuration using namespace options

Elastic Beanstalk provides configuration options for Auto Scaling settings in two namespaces: `aws:autoscaling:asg` and `aws:ec2:instances`.

The aws:autoscaling:asg namespace

The `aws:autoscaling:asg` namespace provides options for overall scale and availability.

The following configuration file example configures the Auto Scaling group to use two to four instances, specific availability zones, and a cooldown period of 12 minutes (720 seconds). It enables Capacity Rebalancing for Spot Instances. This `EnableCapacityRebalancing` option only takes effect if `EnableSpot` is set to `true` in the `aws:ec2:instances` namespace, as shown in the configuration file example following this one.

```
option_settings:
  aws:autoscaling:asg:
    Availability Zones: Any
    Cooldown: '720'
    Custom Availability Zones: 'us-west-2a,us-west-2b'
    MaxSize: '4'
    MinSize: '2'
    EnableCapacityRebalancing: true
```

The aws:ec2:instances namespace

Note

When you update your environment configuration and remove one or more instance types from the `InstanceTypes` option, Elastic Beanstalk terminates any Amazon EC2 instances running on any of the removed instance types. Your environment's Auto Scaling group then launches new instances, as necessary to complete the desired capacity, using your current specified instance types.

The `aws:ec2:instances` namespace provides options related to your environment's instances, including Spot Instance management. It complements `aws:autoscaling:launchconfiguration` and `aws:autoscaling:asg`.

The following configuration file example configures the Auto Scaling group to enable Spot Instance requests for your environment. It designates three possible instance types that can be used. At least one On-Demand Instance is used for baseline capacity, and a sustained 33% of On-Demand Instances is used for any additional capacity.

The configuration sets the spot allocation strategy to `capacity-optimized-prioritized`. This particular allocation strategy prioritizes the instance launches from the pool based on the order of the instance types specified in the `InstanceTypes` option. If `SpotAllocationStrategy` is not specified it defaults to `capacity-optimized`.

```
option_settings:
  aws:ec2:instances:
    EnableSpot: true
    InstanceTypes: 't2.micro,t3.micro,t3.small'
    SpotAllocationStrategy: capacity-optimized-prioritized
    SpotFleetOnDemandBase: '1'
    SpotFleetOnDemandAboveBasePercentage: '33'
```

To choose Spot Instance types, use the Spot Instance Advisor.

Important

The `EnableSpot` option setting can cause Elastic Beanstalk to migrate an existing environment with launch configurations to launch templates. Doing so requires the necessary permissions to manage launch templates. These permissions are included in our managed policy. If you use custom policies instead of our managed policies, environment creation or updates might fail when you update your environment configuration. For more information and other considerations, see Migrating your Elastic Beanstalk environment to launch templates .

Configuration using the AWS CLI

This section provides examples of how you can use the AWS CLI create-environment command to configure your environment with the Auto Scaling and Capacity options described in these sections. You'll notice the namespace settings for `aws:autoscaling:asg` and `aws:ec2:instances`, as described in the previous namespace configuration options section are also configured with this example.

The AWS Command Line Interface provides commands to create and configure Elastic Beanstalk environments. With the `--option-settings` option, you can pass in namespace options that are supported by Elastic Beanstalk. This means that the namespace configuration options described previously can be passed into applicable AWS CLI commands to configure your Elastic Beanstalk environment.

Note

You can also use the update-environment command with `--option-settings` to add or update namespace options. If you need to remove any namespace options from your environment use the **update-environment** command with `--options-to-remove`.

The following example creates a new environment. Refer to the previous topic namespace configuration options for more context about the options that are passed in.

The fist option listed, `IamInstanceProfile` in the aws:autoscaling:launchconfiguration namespace, is the Elastic Beanstalk instance profile. It's required when you create a new environment.

Example — create-environment with Auto Scaling options (namespace options inline)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v4.3.0 running Python 3.12" \
--option-settings \
Namespace=aws:autoscaling:launchconfiguration,OptionName=IamInstanceProfile,Value=aws-
elasticbeanstalk-ec2-role
Namespace=aws:autoscaling:asg,OptionName=Availability Zones,Value=Any \
Namespace=aws:autoscaling:asg,OptionName=Cooldown,Value=720 \
Namespace=aws:autoscaling:asg,OptionName=Custom Availability Zones,Value=us-west-2a,us-
west-2b \
Namespace=aws:autoscaling:asg,OptionName=MaxSize,Value=4 \
Namespace=aws:autoscaling:asg,OptionName=MinSize,Value=2 \
Namespace=aws:autoscaling:asg,OptionName=EnableCapacityRebalancing,Value=true \
Namespace=aws:ec2:instances,OptionName=EnableSpot,Value=true \
Namespace=aws:ec2:instances,OptionName=InstanceTypes,Value=t2.micro,t3.micro,t3.small \
Namespace=aws:ec2:instances,OptionName=SpotAllocationStrategy,Value=capacity-optimized-
prioritized \
Namespace=aws:ec2:instances,OptionName=SpotFleetOnDemandBase,Value=1 \
Namespace=aws:ec2:instances,OptionName=SpotFleetOnDemandAboveBasePercentage,Value=33
```

Important

The `EnableSpot` option setting can cause Elastic Beanstalk to migrate an existing environment with launch configurations to launch templates. Doing so requires the necessary permissions to manage launch templates. These permissions are included in our managed policy. If you use custom policies instead of our managed policies, environment creation or updates might fail when you update your environment configuration. For more information and other considerations, see Migrating your Elastic Beanstalk environment to launch templates .

As an alternative, use an `options.json` file to specify the namespace options instead of including them inline.

Example —create-environment with Auto Scaling options (namespace options in `options.json` file)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v4.3.0 running Python 3.12"
--option-settings file://options.json
```

Example

```
### example options.json ###
[
    {
        "Namespace": "aws:autoscaling:launchconfiguration",
        "OptionName": "IamInstanceProfile",
        "Value": "aws-elasticbeanstalk-ec2-role"
    },
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "Availability Zones",
        "Value": "Any"
    },
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "Cooldown",
        "Value": "720"
    },
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "Custom Availability Zones",
        "Value": "us-west-2a,us-west-2b"
    },
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "MaxSize",
        "Value": "4"
    },
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "MinSize",
        "Value": "2"
    },
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "EnableCapacityRebalancing",
        "Value": "true"
    },
    {
        "Namespace": "aws:ec2:instances",
        "OptionName": "EnableSpot",
        "Value": "true"
    },
    {
        "Namespace": "aws:ec2:instances",
        "OptionName": "InstanceTypes",
        "Value": "t2.micro,t3.micro,t3.small"
    },
    {
        "Namespace": "aws:ec2:instances",
        "OptionName": "SpotAllocationStrategy",
        "Value": "capacity-optimized-prioritized"
    },
    {
        "Namespace": "aws:ec2:instances",
        "OptionName": "SpotFleetOnDemandBase",
        "Value": "1"
    },
    {
        "Namespace": "aws:ec2:instances",
        "OptionName": "SpotFleetOnDemandAboveBasePercentage",
        "Value": "33"
    }
]
```

Configuration using the EB CLI

When creating an environment using the **eb create** command, you can specify a few options that are related to your environment's Auto Scaling group. These are some of the options that help you control the capacity of your environment.

`--single`

Creates the environment with one Amazon EC2 instance and no load balancer. If you don't use this option, a load-balancer is added to the environment that's created.

`--enable-spot`

Enables Spot Instance requests for your environment.

> **Important**
>
> The `enable-spot` option setting can cause Elastic Beanstalk to migrate an existing environment with launch configurations to launch templates. Doing so requires the necessary permissions to manage launch templates. These permissions are included in our managed policy. If you use custom policies instead of our managed policies, environment creation or updates might fail when you update your environment configuration. For more information and other considerations, see Migrating your Elastic Beanstalk environment to launch templates .

The following options for the **eb create** command can only be used with `--enable-spot`.

`--instance-types`

Lists the Amazon EC2 instance types that you want your environment to use.

`--spot-max-price`

The maximum price per unit hour, in US dollars, that you're willing to pay for a Spot Instance. For recommendations about maximum price options for Spot Instances, see Spot Instance pricing history in the *Amazon EC2 User Guide*.

`--on-demand-base-capacity`

The minimum number of On-Demand Instances that your Auto Scaling group provisions before considering Spot Instances as your environment scales up.

`--on-demand-above-base-capacity`

The percentage of On-Demand Instances as part of additional capacity that your Auto Scaling group provisions that's more than the number of instances that's specified by the `--on-demand-base-capacity` option.

The following example creates an environment and configures the Auto Scaling group to enable Spot Instance requests for the new environment. For this example, three possible instance types can be used.

```
$ eb create --enable-spot --instance-types "t2.micro,t3.micro,t3.small"
```

Important

There is another similarly named option that's called `--instance-type` (no “s”) that the EB CLI only recognizes when processing On-Demand Instances. Don't use `--instance-type` (no "s") with the `--enable-spot` option. If you do, the EB CLI ignores it. Instead use `--instance-types` (with "s") with the `--enable-spot` option.

Auto Scaling triggers for your Elastic Beanstalk environment

The Auto Scaling group in your Elastic Beanstalk environment uses two Amazon CloudWatch alarms to trigger scaling operations. The default triggers scale when the average outbound network traffic from each instance is higher than 6 MB or lower than 2 MB over a period of five minutes. To use Amazon EC2 Auto Scaling effectively, configure triggers that are appropriate for your application, instance type, and service requirements. You can scale based on several statistics including latency, disk I/O, CPU utilization, and request count.

For more information about CloudWatch metrics and alarms, see Amazon CloudWatch Concepts in the *Amazon CloudWatch User Guide*.

Configuring Auto Scaling triggers

You can configure the triggers that adjust the number of instances in your environment's Auto Scaling group in the Elastic Beanstalk console.

To configure triggers in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. In the **Scaling triggers** section, configure the following settings:

   - **Metric** – Metric used for your Auto Scaling trigger.

- **Statistic** – Statistic calculation the trigger should use, such as `Average`.
- **Unit** – Unit for the trigger metric, such as **Bytes**.
- **Period** – Specifies how frequently Amazon CloudWatch measures the metrics for your trigger.
- **Breach duration** – Amount of time, in minutes, a metric can be outside of the upper and lower thresholds before triggering a scaling operation.
- **Upper threshold** – If the metric exceeds this number for the breach duration, a scaling operation is triggered.
- **Scale up increment** – The number of Amazon EC2 instances to add when performing a scaling activity.
- **Lower threshold** – If the metric falls below this number for the breach duration, a scaling operation is triggered.
- **Scale down increment** – The number of Amazon EC2 instances to remove when performing a scaling activity.

6. To save the changes choose **Apply** at the bottom of the page.

The aws:autoscaling:trigger namespace

Elastic Beanstalk provides configuration options for Auto Scaling settings in the `aws:autoscaling:trigger` namespace. Settings in this namespace are organized by the resource that they apply to.

```
option_settings:
  AWSEBAutoScalingScaleDownPolicy.aws:autoscaling:trigger:
    LowerBreachScaleIncrement: '-1'
  AWSEBAutoScalingScaleUpPolicy.aws:autoscaling:trigger:
    UpperBreachScaleIncrement: '1'
  AWSEBCloudwatchAlarmHigh.aws:autoscaling:trigger:
    UpperThreshold: '6000000'
  AWSEBCloudwatchAlarmLow.aws:autoscaling:trigger:
    BreachDuration: '5'
    EvaluationPeriods: '1'
    LowerThreshold: '2000000'
    MeasureName: NetworkOut
    Period: '5'
    Statistic: Average
    Unit: Bytes
```

Scheduled Auto Scaling actions for your Elastic Beanstalk environments

To optimize your environment's use of Amazon EC2 instances through predictable periods of peak traffic, configure your Amazon EC2 Auto Scaling group to change its instance count on a schedule. You can configure your environment with a recurring action to scale up each day in the morning, and scale down at night when traffic is low. For example, if you have a marketing event that will drive traffic to your site for a limited period of time, you can schedule a one-time event to scale up when it starts, and another to scale down when it ends.

You can define up to 120 active scheduled actions per environment. Elastic Beanstalk also retains up to 150 expired scheduled actions, which you can reuse by updating their settings.

Configuring scheduled actions

You can create scheduled actions for your environment's Auto Scaling group in the Elastic Beanstalk console.

To configure scheduled actions in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. In the **Time-based scaling** section, choose **Add scheduled action**.
6. Fill in the following scheduled action settings:

   - **Name –** Specify a unique name of up to 255 alphanumeric characters, with no spaces.
   - **Instances –** Choose the minimum and maximum instance count to apply to the Auto Scaling group.
   - **Desired capacity** (optional) – Set the initial desired capacity for the Auto Scaling group. After the scheduled action is applied, triggers adjust the desired capacity based on their settings.
   - **Occurrence –** Choose **Recurring** to repeat the scaling action on a schedule.
   - **Start time –** For one-time actions, choose the date and time to run the action.

For recurrent actions, a start time is optional. Specify it to choose the earliest time the action is performed. After this time, the action recurs according to the **Recurrence** expression.

- **Recurrence** – Use a Cron expression to specify the frequency with which you want the scheduled action to occur. For example, `30 6 * * 2` runs the action every Tuesday at 6:30 AM UTC.
- **End time** (optional) – Optional for recurrent actions. If specified, the action recurs according to the **Recurrence** expression, and is not performed again after this time.

  When a scheduled action ends, Auto Scaling doesn't automatically go back to its previous settings. Configure a second scheduled action to return Auto Scaling to the original settings as needed.

7. Choose **Add**.
8. To save the changes choose **Apply** at the bottom of the page.

> **Note**
>
> Scheduled actions will not be saved until applied.

The aws:autoscaling:scheduledaction namespace

If you need to configure a large number of scheduled actions, you can use configuration files or the Elastic Beanstalk API to apply the configuration option changes from a YAML or JSON file. These methods also let you access the Suspend option to temporarily deactivate a recurrent scheduled action.

> **Note**
>
> When working with scheduled action configuration options outside of the console, use ISO 8601 time format to specify start and end times in UTC. For example, 2015-04-28T04:07:02Z. For more information about ISO 8601 time format, see Date and Time Formats. The dates must be unique across all scheduled actions.

Elastic Beanstalk provides configuration options for scheduled action settings in the `aws:autoscaling:scheduledaction` namespace. Use the `resource_name` field to specify the name of the scheduled action.

Example Scheduled-scale-up-specific-time-long.config

This configuration file instructs Elastic Beanstalk to scale out from five instances to 10 instances at 2015-12-12T00:00:00Z.

```
option_settings:
  - namespace: aws:autoscaling:scheduledaction
    resource_name: ScheduledScaleUpSpecificTime
    option_name: MinSize
    value: '5'
  - namespace: aws:autoscaling:scheduledaction
    resource_name: ScheduledScaleUpSpecificTime
    option_name: MaxSize
    value: '10'
  - namespace: aws:autoscaling:scheduledaction
    resource_name: ScheduledScaleUpSpecificTime
    option_name: DesiredCapacity
    value: '5'
  - namespace: aws:autoscaling:scheduledaction
    resource_name: ScheduledScaleUpSpecificTime
    option_name: StartTime
    value: '2015-12-12T00:00:00Z'
```

Example Scheduled-scale-up-specific-time.config

To use the shorthand syntax with the EB CLI or configuration files, prepend the resource name to the namespace.

```
option_settings:
  ScheduledScaleUpSpecificTime.aws:autoscaling:scheduledaction:
    MinSize: '5'
    MaxSize: '10'
    DesiredCapacity: '5'
    StartTime: '2015-12-12T00:00:00Z'
```

Example Scheduled-scale-down-specific-time.config

This configuration file instructs Elastic Beanstalk to scale in at 2015-12-12T07:00:00Z.

```
option_settings:
  ScheduledScaleDownSpecificTime.aws:autoscaling:scheduledaction:
    MinSize: '1'
    MaxSize: '1'
    DesiredCapacity: '1'
    StartTime: '2015-12-12T07:00:00Z'
```

Example Scheduled-periodic-scale-up.config

This configuration file instructs Elastic Beanstalk to scale out every day at 9AM. The action is scheduled to begin May 14, 2015 and end January 12, 2016.

```
option_settings:
  ScheduledPeriodicScaleUp.aws:autoscaling:scheduledaction:
    MinSize: '5'
    MaxSize: '10'
    DesiredCapacity: '5'
    StartTime: '2015-05-14T07:00:00Z'
    EndTime: '2016-01-12T07:00:00Z'
    Recurrence: 0 9 * * *
```

Example Scheduled-periodic-scale-down.config

This configuration file instructs Elastic Beanstalk to scale in to no running instance every day at 6PM. If you know that your application is mostly idle outside of business hours, you can create a similar scheduled action. If your application must be down outside of business hours, change `MaxSize` to `0`.

```
option_settings:
  ScheduledPeriodicScaleDown.aws:autoscaling:scheduledaction:
    MinSize: '0'
    MaxSize: '1'
    DesiredCapacity: '0'
    StartTime: '2015-05-14T07:00:00Z'
    EndTime: '2016-01-12T07:00:00Z'
    Recurrence: 0 18 * * *
```

Example Scheduled-weekend-scale-down.config

This configuration file instructs Elastic Beanstalk to scale in every Friday at 6PM. If you know that your application doesn’t receive as much traffic over the weekend, you can create a similar scheduled action.

```
option_settings:
  ScheduledWeekendScaleDown.aws:autoscaling:scheduledaction:
    MinSize: '1'
    MaxSize: '4'
    DesiredCapacity: '1'
    StartTime: '2015-12-12T07:00:00Z'
    EndTime: '2016-01-12T07:00:00Z'
    Recurrence: 0 18 * * 5
```

Auto Scaling health check setting for your Elastic Beanstalk environment

Amazon EC2 Auto Scaling monitors the health of each Amazon Elastic Compute Cloud (Amazon EC2) instance that it launches. If any instance terminates unexpectedly, Auto Scaling detects the termination and launches a replacement instance. By default, the Auto Scaling group created for your environment uses Amazon EC2 status checks. If an instance in your environment fails an Amazon EC2 status check, Auto Scaling takes it down and replaces it.

Amazon EC2 status checks only cover an instance's health, not the health of your application, server, or any Docker containers running on the instance. If your application crashes, but the instance that it runs on is still healthy, it may be kicked out of the load balancer, but Auto Scaling won't replace it automatically. The default behavior is good for troubleshooting. If Auto Scaling replaced the instance as soon as the application crashed, you might not realize that anything went wrong, even if it crashed quickly after starting up.

If you want Auto Scaling to replace instances whose application has stopped responding, you can use a configuration file to configure the Auto Scaling group to use Elastic Load Balancing health checks. The following example sets the group to use the load balancer's health checks, in addition to the Amazon EC2 status check, to determine an instance's health.

Example .ebextensions/autoscaling.config

```
Resources:
  AWSEBAutoScalingGroup:
    Type: "AWS::AutoScaling::AutoScalingGroup"
    Properties:
      HealthCheckType: ELB
      HealthCheckGracePeriod: 300
```

For more information about the `HealthCheckType` and `HealthCheckGracePeriod` properties, see AWS::AutoScaling::AutoScalingGroup in the *AWS CloudFormation User Guide* and Health Checks for Auto Scaling Instances in the *Amazon EC2 Auto Scaling User Guide*.

By default, the Elastic Load Balancing health check is configured to attempt a TCP connection to your instance over port 80. This confirms that the web server running on the instance is accepting connections. However, you might want to customize the load balancer health check to ensure that your application, and not just the web server, is in a good state. The grace period setting sets the number of seconds that an instance can fail the health check without being terminated and replaced. Instances can recover after being kicked out of the load balancer, so give the instance an amount of time that is appropriate for your application.

Load balancer for your Elastic Beanstalk environment

A load balancer distributes traffic among your environment's instances. When you enable load balancing, AWS Elastic Beanstalk creates an Elastic Load Balancing load balancer dedicated to your environment. Elastic Beanstalk fully manages this load balancer, taking care of security settings and of terminating the load balancer when you terminate your environment.

Alternatively, you can choose to share a load balancer across several Elastic Beanstalk environments. With a shared load balancer, you save on operational cost by avoiding a dedicated load balancer for each environment. You also assume more of the management responsibility for the shared load balancer that your environments use.

Elastic Load Balancing has these load balancer types:

- Classic Load Balancer – The previous-generation load balancer. Routes HTTP, HTTPS, or TCP request traffic to different ports on environment instances.
- Application Load Balancer – An application layer load balancer. Routes HTTP or HTTPS request traffic to different ports on environment instances based on the request path.
- Network Load Balancer – A network layer load balancer. Routes TCP request traffic to different ports on environment instances. Supports both active and passive health checks.

Elastic Beanstalk supports all three load balancer types. The following table shows which types you can use with the two usage patterns:

Load balancer type	Dedicated	Shared
Classic Load Balancer	✓ Yes	✗ No
Application Load Balancer	✓ Yes	✓ Yes
Network Load Balancer	✓ Yes	✗ No

Note

The Classic Load Balancer (CLB) option is disabled on the **Create Environment** console wizard. If you have an existing environment configured with a Classic Load Balancer you can create a new one by cloning the existing environment using either the Elastic Beanstalk console or the EB CLI. You also have the option to use the EB CLI or the AWS CLI to create a new environment configured with a Classic Load Balancer. These command line tools will create a new environment with a CLB even if one doesn't already exist in your account.

By default, Elastic Beanstalk creates an Application Load Balancer for your environment when you enable load balancing with the Elastic Beanstalk console or the EB CLI. It configures the load balancer to listen for HTTP traffic on port 80 and forward this traffic to instances on the same port. You can choose the type of load balancer that your environment uses only during environment creation. Later, you can change settings to manage the behavior of your running environment's load balancer, but you can't change its type.

Note

Your environment must be in a VPC with subnets in at least two Availability Zones to create an Application Load Balancer. All new AWS accounts include default VPCs that meet this requirement.

See the following topics to learn about each load balancer type that Elastic Beanstalk supports, its functionality, how to configure and manage it in an Elastic Beanstalk environment, and how to configure a load balancer to upload access logs to Amazon S3.

Topics

Configuring a Classic Load Balancer

When you enable load balancing, your AWS Elastic Beanstalk environment is equipped with an Elastic Load Balancing load balancer to distribute traffic among the instances in your environment. Elastic Load Balancing supports several load balancer types. To learn about them, see the Elastic Load Balancing User Guide. Elastic Beanstalk can create a load balancer for you, or let you specify a shared load balancer that you've created.

This topic describes the configuration of a Classic Load Balancer that Elastic Beanstalk creates and dedicates to your environment. For information about configuring all the load balancer types that Elastic Beanstalk supports, see Load balancer for your Elastic Beanstalk environment.

> **Note**
>
> You can choose the type of load balancer that your environment uses only during environment creation. Later, you can change settings to manage the behavior of your running environment's load balancer, but you can't change its type.

Introduction

A Classic Load Balancer is the Elastic Load Balancing previous-generation load balancer. It supports routing HTTP, HTTPS, or TCP request traffic to different ports on environment instances.

When your environment uses a Classic Load Balancer, Elastic Beanstalk configures it by default to listen for HTTP traffic on port 80 and forward it to instances on the same port. Although you cannot delete the port 80 default listener, you can disable it, which achieves the same functionality by blocking traffic. Note that you can add or delete other listeners. To support secure connections, you can configure your load balancer with a listener on port 443 and a TLS certificate.

The load balancer uses a health check to determine whether the Amazon EC2 instances running your application are healthy. The health check makes a request to a specified URL at a set interval. If the URL returns an error message, or fails to return within a specified timeout period, the health check fails.

If your application performs better by serving multiple requests from the same client on a single server, you can configure your load balancer to use sticky sessions. With sticky sessions, the load balancer adds a cookie to HTTP responses that identifies the Amazon EC2 instance that served the request. When a subsequent request is received from the same client, the load balancer uses the cookie to send the request to the same instance.

With cross-zone load balancing, each load balancer node for your Classic Load Balancer distributes requests evenly across the registered instances in all enabled Availability Zones. If cross-zone load balancing is disabled, each load balancer node distributes requests evenly across the registered instances in its Availability Zone only.

When an instance is removed from the load balancer because it has become unhealthy or the environment is scaling down, connection draining gives the instance time to complete requests before closing the connection between the instance and the load balancer. You can change the amount of time given to instances to send a response, or disable connection draining completely.

Note

Connection draining is enabled by default when you create an environment with the Elastic Beanstalk console or the EB CLI. For other clients, you can enable it with configuration options.

You can use advanced load balancer settings to configure listeners on arbitrary ports, modify additional sticky session settings, and configure the load balancer to connect to EC2 instances securely. These settings are available through configuration options that you can set by using configuration files in your source code, or directly on an environment by using the Elastic Beanstalk API. Many of these settings are also available in the Elastic Beanstalk console. In addition, you can configure a load balancer to upload access logs to Amazon S3.

Configuring a Classic Load Balancer using the Elastic Beanstalk console

You can use the Elastic Beanstalk console to configure a Classic Load Balancer's ports, HTTPS certificate, and other settings, during environment creation or later when your environment is running.

> **Note**
>
> The Classic Load Balancer (CLB) option is disabled on the **Create Environment** console wizard. If you have an existing environment configured with a Classic Load Balancer you can create a new one by cloning the existing environment using either the Elastic Beanstalk console or the EB CLI. You also have the option to use the EB CLI or the AWS CLI to create a new environment configured with a Classic Load Balancer. These command line tools will create a new environment with a CLB even if one doesn't already exist in your account.

To configure a running environment's Classic Load Balancer in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Load balancer** configuration category, choose **Edit**.

   > **Note**
   >
   > If the **Load balancer** configuration category doesn't have an **Edit** button, your environment doesn't have a load balancer. To learn how to set one up, see Changing environment type.

5. Make the Classic Load Balancer configuration changes that your environment requires.
6. To save the changes choose **Apply** at the bottom of the page.

Classic Load Balancer settings

- Listeners
- Sessions

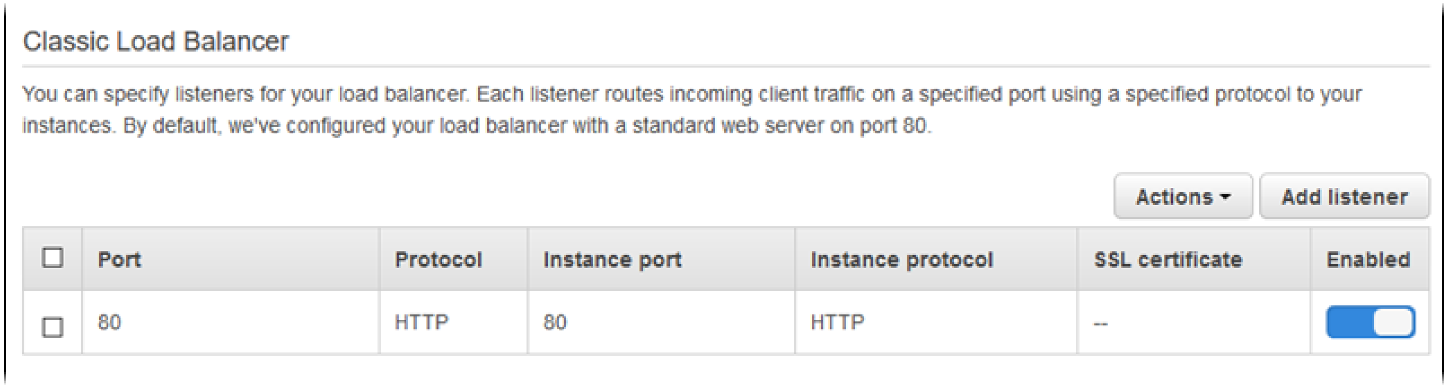
- Cross-zone load balancing
- Connection draining
- Health check

Listeners

Use this list to specify listeners for your load balancer. Each listener routes incoming client traffic on a specified port using a specified protocol to your instances. Initially, the list shows the default listener, which routes incoming HTTP traffic on port 80 to your environment's instance servers that are listening to HTTP traffic on port 80.

Note

Although you cannot delete the port 80 default listener, you can disable it, which achieves the same functionality by blocking traffic.



Classic Load Balancer

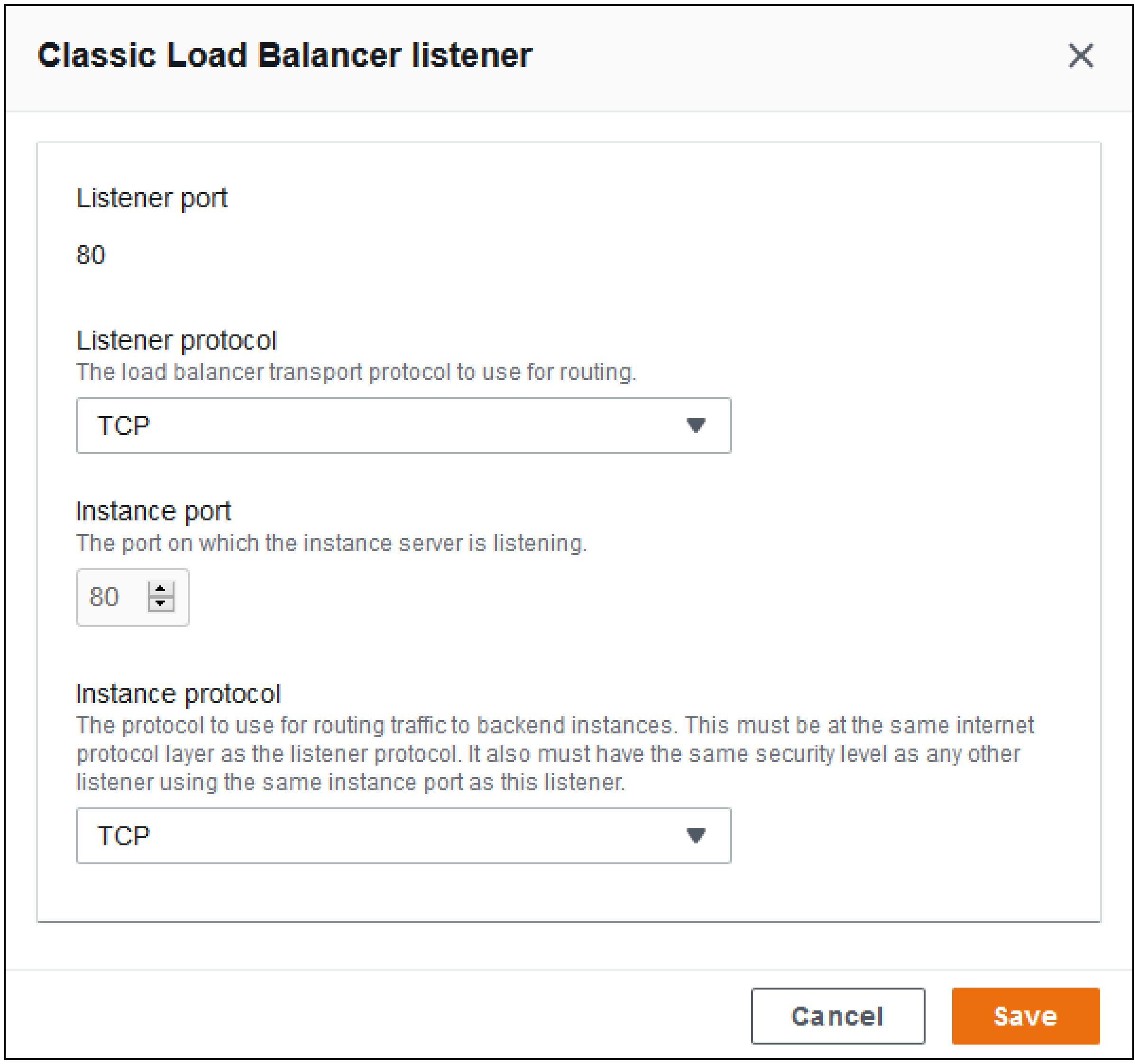
You can specify listeners for your load balancer. Each listener routes incoming client traffic on a specified port using a specified protocol to your instances. By default, we've configured your load balancer with a standard web server on port 80.

☐	Port	Protocol	Instance port	Instance protocol	SSL certificate	Enabled
☐	80	HTTP	80	HTTP	--	✓

To configure an existing listener

1. Select the check box next to its table entry, choose **Actions**, and then choose the action you want.
2. If you chose **Edit**, use the **Classic Load Balancer listener** dialog box to edit settings, and then choose **Save**.

For example, you can edit the default listener and change the **Protocol** from **HTTP** to **TCP** if you want the load balancer to forward a request as is. This prevents the load balancer from rewriting headers (including `X-Forwarded-For`). The technique doesn't work with sticky sessions.


Classic Load Balancer listener

Listener port

80

Listener protocol
The load balancer transport protocol to use for routing.

TCP

Instance port
The port on which the instance server is listening.

80

Instance protocol
The protocol to use for routing traffic to backend instances. This must be at the same internet protocol layer as the listener protocol. It also must have the same security level as any other listener using the same instance port as this listener.

TCP

To add a listener

1. Choose **Add listener**.
2. In the **Classic Load Balancer listener** dialog box, configure the settings you want, and then choose **Add**.

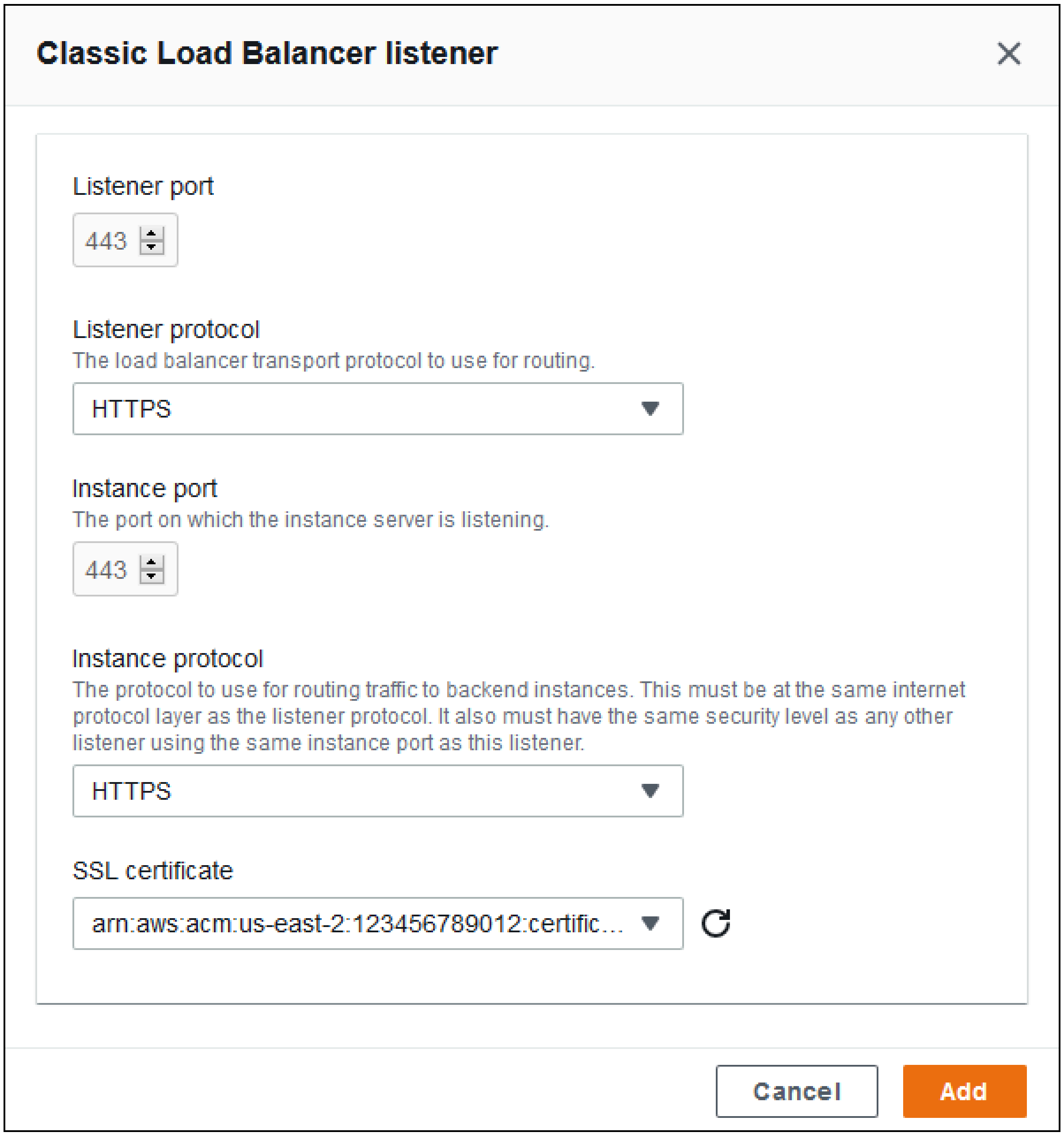
Adding a secure listener is a common use case. The example in the following image adds a listener for HTTPS traffic on port 443. This listener routes the incoming traffic to environment instance servers listening to HTTPS traffic on port 443.

Before you can configure an HTTPS listener, ensure that you have a valid SSL certificate. Do one of the following:

- If AWS Certificate Manager (ACM) is available in your AWS Region, create or import a certificate using ACM. For more information about requesting an ACM certificate, see Request a Certificate in the *AWS Certificate Manager User Guide*. For more information about importing third-party certificates into ACM, see Importing Certificates in the *AWS Certificate Manager User Guide*.
- If ACM isn't available in your AWS Region, upload your existing certificate and key to IAM. For more information about creating and uploading certificates to IAM, see Working with Server Certificates in the *IAM User Guide*.

For more detail on configuring HTTPS and working with certificates in Elastic Beanstalk, see Configuring HTTPS for your Elastic Beanstalk environment.

For **SSL certificate**, choose the ARN of your SSL certificate. For example, `arn:aws:iam::123456789012:server-certificate/abc/certs/build`, or `arn:aws:acm:us-east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678`.


Classic Load Balancer listener

Listener port

443

Listener protocol

The load balancer transport protocol to use for routing.

HTTPS

Instance port

The port on which the instance server is listening.

443

Instance protocol

The protocol to use for routing traffic to backend instances. This must be at the same internet protocol layer as the listener protocol. It also must have the same security level as any other listener using the same instance port as this listener.

HTTPS

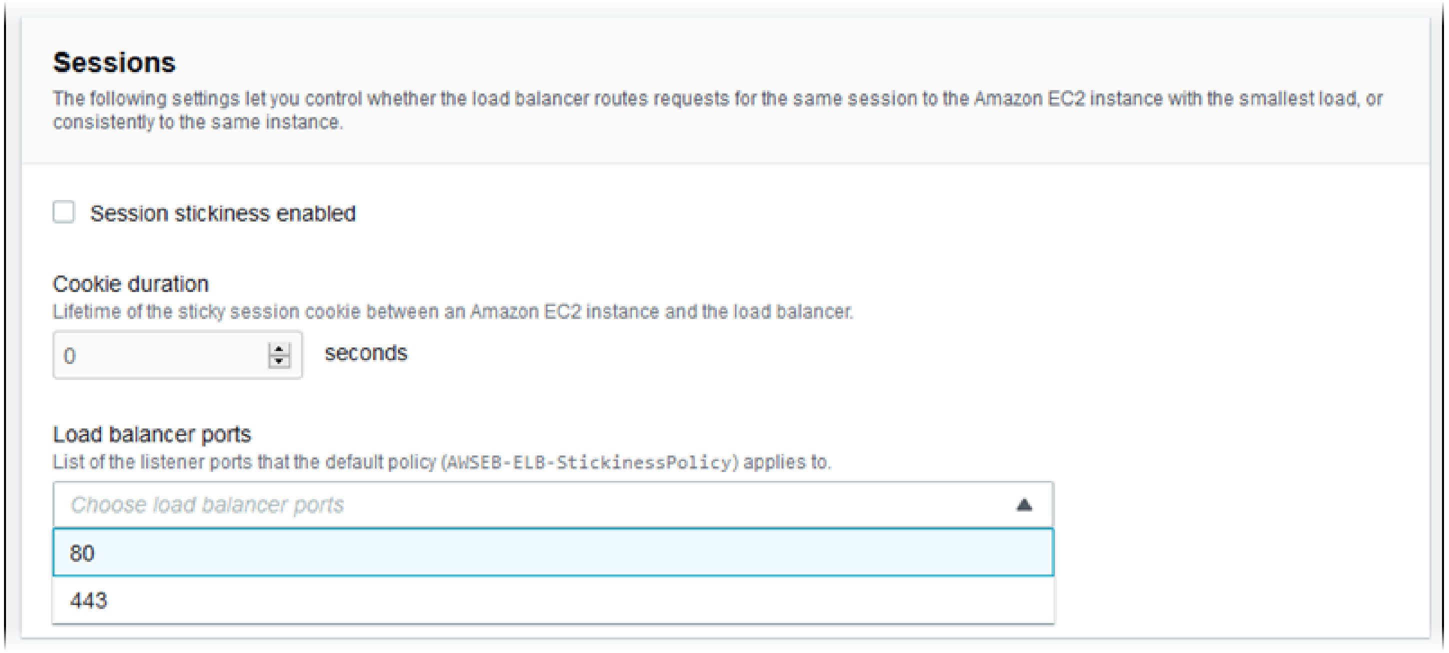
SSL certificate

arn:aws:acm:us-east-2:123456789012:certific...

For details about configuring HTTPS and working with certificates in Elastic Beanstalk, see Configuring HTTPS for your Elastic Beanstalk environment.

Sessions

Select or clear the **Session stickiness enabled** box to enable or disable sticky sessions. Use **Cookie duration** to configure a sticky session's duration, up to **1000000** seconds. On the **Load balancer ports** list, select listener ports that the default policy (`AWSEB-ELB-StickinessPolicy`) applies to.


Sessions

The following settings let you control whether the load balancer routes requests for the same session to the Amazon EC2 instance with the smallest load, or consistently to the same instance.

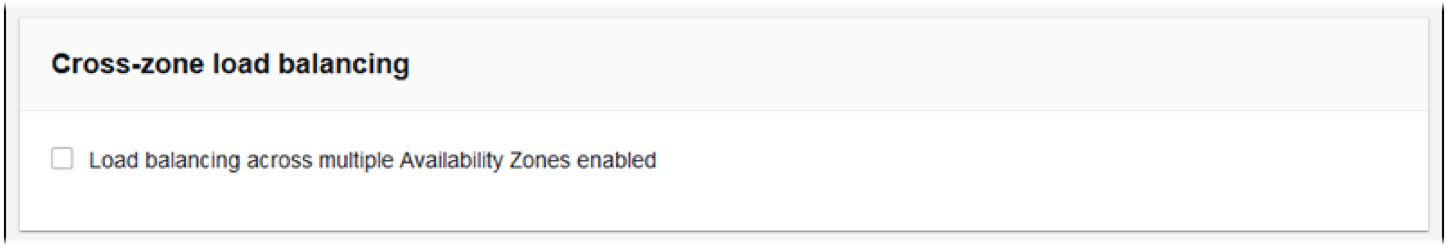
☐ Session stickiness enabled

Cookie duration
Lifetime of the sticky session cookie between an Amazon EC2 instance and the load balancer.
0 seconds

Load balancer ports
List of the listener ports that the default policy (`AWSEB-ELB-StickinessPolicy`) applies to.
Choose load balancer ports
80
443

Cross-zone load balancing

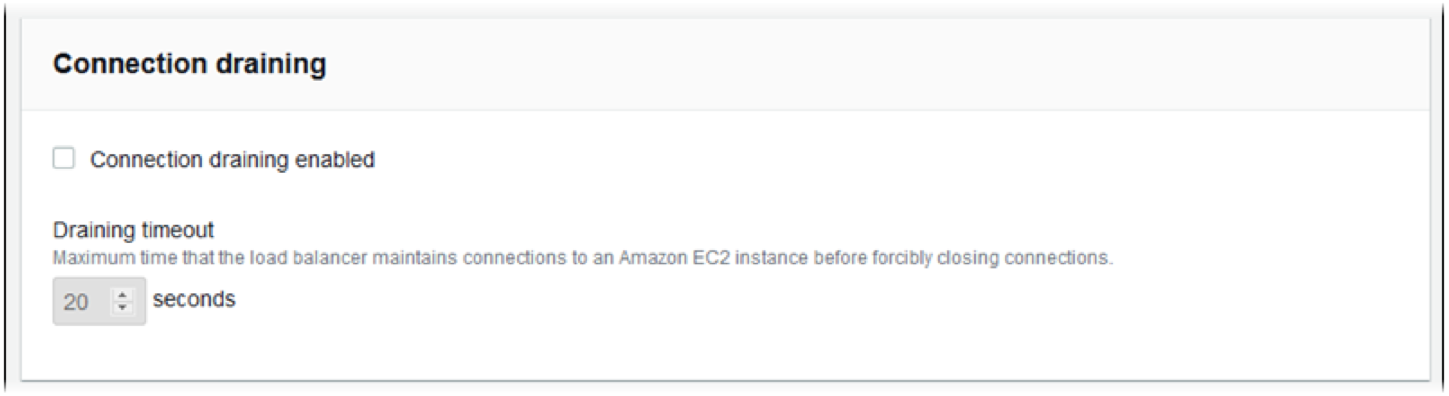
Select or clear the **Load balancing across multiple Availability Zones enabled** box to enable or disable cross-zone load balancing.


Cross-zone load balancing

☐ Load balancing across multiple Availability Zones enabled

Connection draining

Select or clear the **Connection draining enabled** box to enable or disable connection draining. Set the **Draining timeout**, up to **`3600`** seconds.


Connection draining

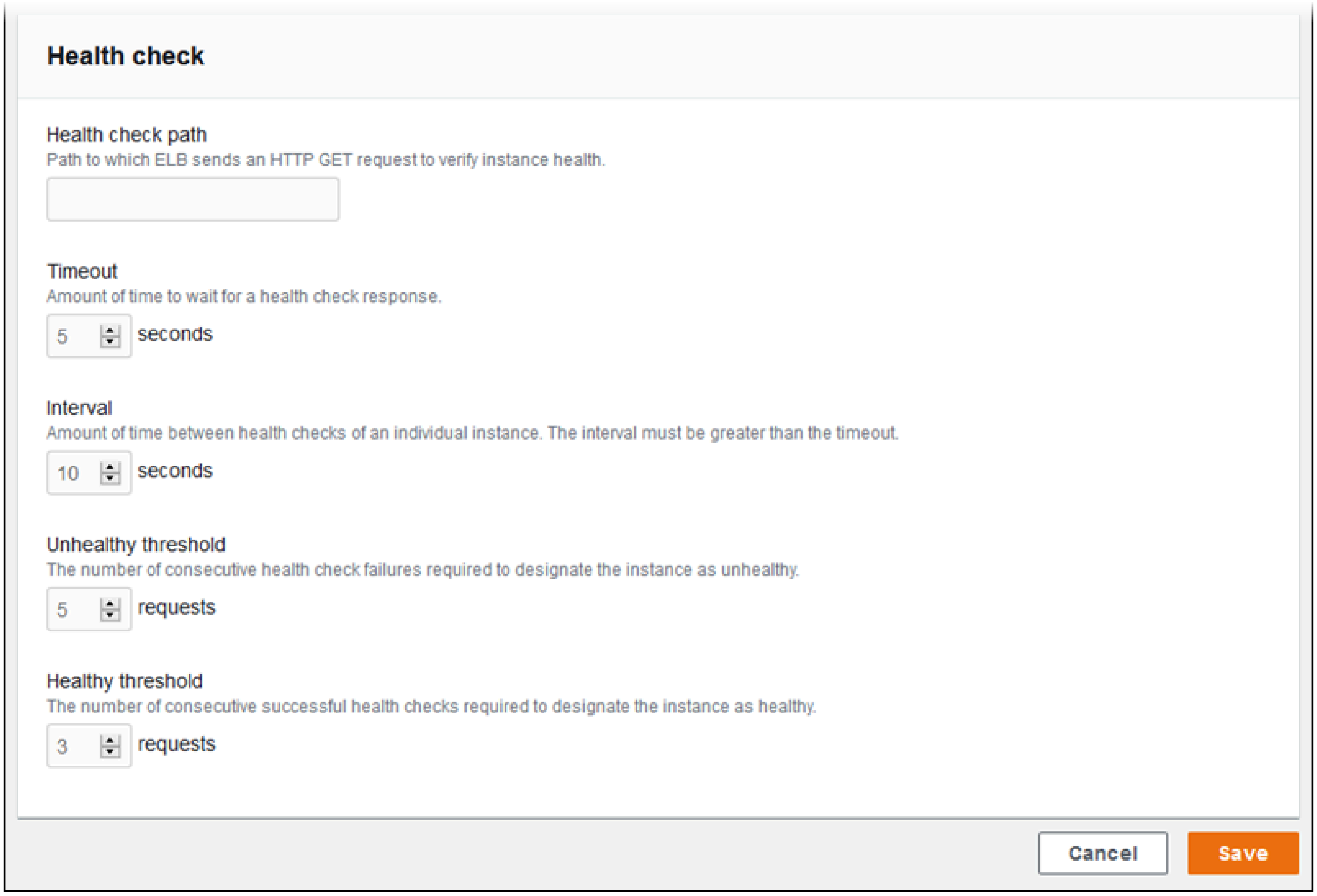
☐ Connection draining enabled

Draining timeout
Maximum time that the load balancer maintains connections to an Amazon EC2 instance before forcibly closing connections.
20 seconds

Health check

Use the following settings to configure load balancer health checks:

- **Health check path** – The path to which the load balancer sends health check requests. If you don't set the path, the load balancer attempts to make a TCP connection on port 80 to verify health.
- **Timeout** – The amount of time, in seconds, to wait for a health check response.
- **Interval** – The amount of time, in seconds, between health checks of an individual instance. The interval must be greater than the timeout.
- **Unhealthy threshold**, **Healthy threshold** – The number of health checks that must fail or pass, respectively, before Elastic Load Balancing changes an instance's health state.



Health check

Health check path
Path to which ELB sends an HTTP GET request to verify instance health.

Timeout
Amount of time to wait for a health check response.
5 seconds

Interval
Amount of time between health checks of an individual instance. The interval must be greater than the timeout.
10 seconds

Unhealthy threshold
The number of consecutive health check failures required to designate the instance as unhealthy.
5 requests

Healthy threshold
The number of consecutive successful health checks required to designate the instance as healthy.
3 requests

Note

The Elastic Load Balancing health check doesn't affect the health check behavior of an environment's Auto Scaling group. Instances that fail an Elastic Load Balancing health check are not automatically replaced by Amazon EC2 Auto Scaling unless you manually configure Amazon EC2 Auto Scaling to do so. See Auto Scaling health check setting for your Elastic Beanstalk environment for details.

For more information about health checks and how they influence your environment's overall health, see Basic health reporting.

Configuring a Classic Load Balancer using the EB CLI

The EB CLI prompts you to choose a load balancer type when you run **eb create**.

```
$ eb create
Enter Environment Name
(default is my-app): test-env
Enter DNS CNAME prefix
(default is my-app): test-env-DLW24ED23SF

Select a load balancer type
1) classic
2) application
3) network
(default is 1):
```

Press **Enter** to select `classic`.

You can also specify a load balancer type by using the `--elb-type` option.

```
$ eb create test-env --elb-type classic
```

Classic Load Balancer configuration namespaces

You can find settings related to Classic Load Balancers in the following namespaces:

- `aws:elb:healthcheck` – Configure the thresholds, check interval, and timeout for load balancer health checks.

- `aws:elasticbeanstalk:application` – Configure the health check URL.
- `aws:elb:loadbalancer` – Enable cross-zone load balancing. Assign security groups to the load balancer and override the default security group that Elastic Beanstalk creates. This namespace also includes deprecated options for configuring the standard and secure listeners that have been replaced by options in the `aws:elb:listener` namespace.
- `aws:elb:listener` – Configure the default listener on port 80, a secure listener on port 443, or additional listeners for any protocol on any port. If you specify `aws:elb:listener` as the namespace, settings apply to the default listener on port 80. If you specify a port (for example, `aws:elb:listener:443`), a listener is configured on that port.
- `aws:elb:policies` – Configure additional settings for your load balancer. Use options in this namespace to configure listeners on arbitrary ports, modify additional sticky session settings, and configure the load balancer to connect to Amazon EC2 instances securely.

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See Recommended values for details.

Example .ebextensions/loadbalancer-terminatehttps.config

The following example configuration file creates an HTTPS listener on port 443, assigns a certificate that the load balancer uses to terminate the secure connection, and disables the default listener on port 80. The load balancer forwards the decrypted requests to the EC2 instances in your environment on HTTP:80.

```
option_settings:
  aws:elb:listener:443:
    ListenerProtocol: HTTPS
    SSLCertificateId: arn:aws:acm:us-
east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678
    InstancePort: 80
    InstanceProtocol: HTTP
  aws:elb:listener:
    ListenerEnabled: false
```

Configuring an Application Load Balancer

When you enable load balancing, your AWS Elastic Beanstalk environment is equipped with an Elastic Load Balancing load balancer to distribute traffic among the instances in your environment.

Elastic Load Balancing supports several load balancer types. To learn about them, see the Elastic Load Balancing User Guide. Elastic Beanstalk can create a load balancer for you, or let you specify a shared load balancer that you've created.

This topic describes the configuration of an Application Load Balancer that Elastic Beanstalk creates and dedicates to your environment. See also the section called “Shared Application Load Balancer”. For information about configuring all the load balancer types that Elastic Beanstalk supports, see the section called “Load balancer”.

Note

You can choose the type of load balancer that your environment uses only during environment creation. You can change settings to manage the behavior of your running environment's load balancer, but you can't change its type. You also can't switch from a dedicated to a shared load balancer or vice versa.

Introduction

An Application Load Balancer inspects traffic at the application network protocol layer to identify the request's path so that it can direct requests for different paths to different destinations.

When your environment uses an Application Load Balancer, Elastic Beanstalk configures it by default to perform the same function as a Classic Load Balancer. The default listener accepts HTTP requests on port 80 and distributes them to the instances in your environment. You can add a secure listener on port 443 with a certificate to decrypt HTTPS traffic, configure health check behavior, and push access logs from the load balancer to an Amazon Simple Storage Service (Amazon S3) bucket.

Note

Unlike a Classic Load Balancer or a Network Load Balancer, an Application Load Balancer can't have transport layer (layer 4) TCP or SSL/TLS listeners. It supports only HTTP and HTTPS listeners. Additionally, it can't use backend authentication to authenticate HTTPS connections between the load balancer and backend instances.

In an Elastic Beanstalk environment, you can use an Application Load Balancer to direct traffic for certain paths to a different process on your web server instances. With a Classic Load Balancer, all traffic to a listener is routed to a single process on the backend instances. With an Application Load Balancer, you can configure multiple *rules* on the listener to route requests to certain paths to different backend process. You configure each process with the port that the process listens on.

For example, you could run a login process separately from your main application. While the main application on your environment's instances accepts the majority of requests and listens on port 80, your login process listens on port 5000 and accepts requests to the `/login` path. All incoming requests from clients come in on port 80. With an Application Load Balancer, you can configure a single listener for incoming traffic on port 80, with two rules that route traffic to two separate processes, depending on the path in the request. You add a custom rule that routes traffic to `/login` to the login process listening on port 5000. The default rule routes all other traffic to the main application process listening on port 80.

An Application Load Balancer rule maps a request to a *target group*. In Elastic Beanstalk, a target group is represented by a *process*. You can configure a process with a protocol, port, and health check settings. The process represents the process running on the instances in your environment. The default process is a listener on port 80 of the reverse proxy (nginx or Apache) that runs in front of your application.

Note

Outside of Elastic Beanstalk, a target group maps to a group of instances. A listener can use rules and target groups to route traffic to different instances based on the path. Within Elastic Beanstalk, all of your instances in your environment are identical, so the distinction is made between processes listening on different ports.

A Classic Load Balancer uses a single health check path for the entire environment. With an Application Load Balancer, each process has a separate health check path that is monitored by the load balancer and Elastic Beanstalk-enhanced health monitoring.

To use an Application Load Balancer, your environment must be in a default or custom VPC, and must have a service role with the standard set of permissions. If you have an older service role, you might need to update the permissions on it to include `elasticloadbalancing:DescribeTargetHealth` and `elasticloadbalancing:DescribeLoadBalancers`. For more information about Application Load Balancers, see What is an Application Load Balancer?.

Note

The Application Load Balancer health check doesn't use the Elastic Beanstalk health check path. Instead, it uses the specific path configured for each process separately.

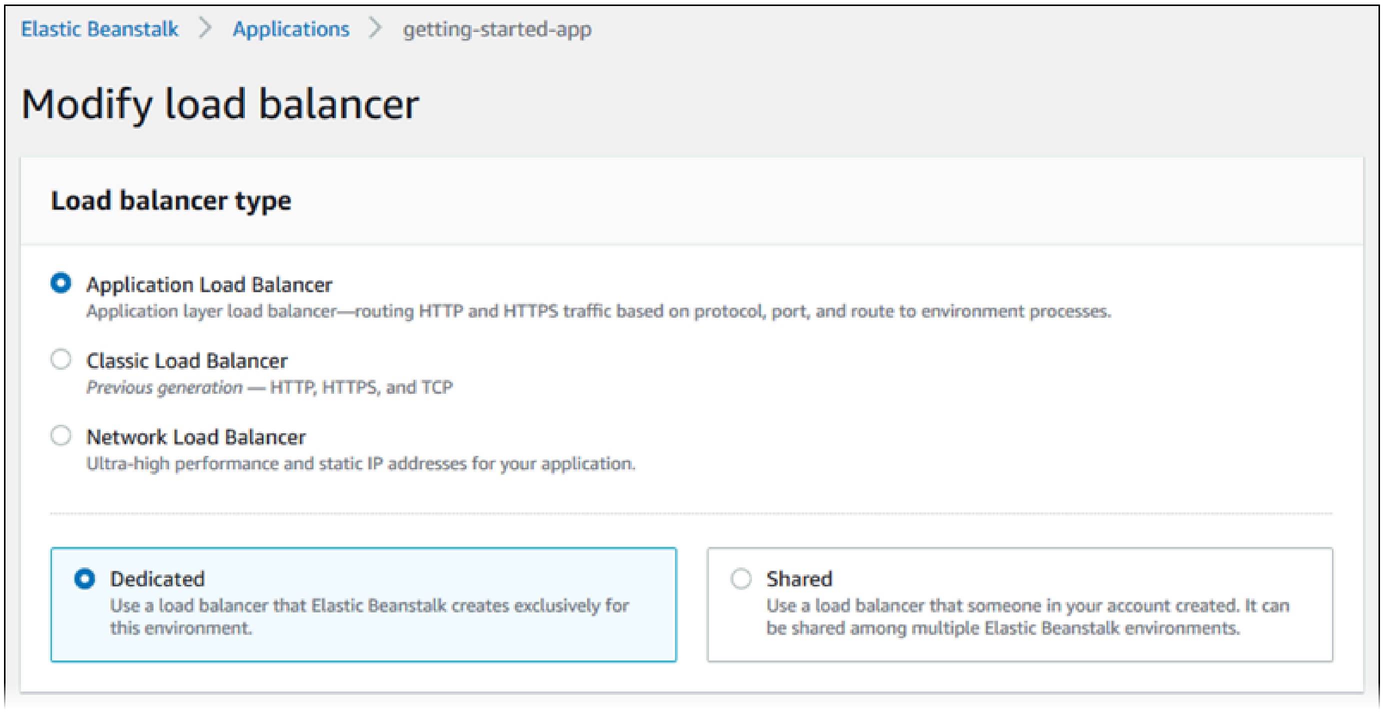
Configuring an Application Load Balancer using the Elastic Beanstalk console

You can use the Elastic Beanstalk console to configure an Application Load Balancer's listeners, processes, and rules, during environment creation or later when your environment is running.

To configure an Application Load Balancer in the Elastic Beanstalk console during environment creation

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**.
3. Choose Create a new environment to start creating your environment.
4. On the wizard's main page, before choosing **Create environment**, choose **Configure more options**.
5. Choose the **High availability** configuration preset.

   Alternatively, in the **Capacity** configuration category, configure a **Load balanced** environment type. For details, see Capacity.
6. In the **Load balancer** configuration category, choose **Edit**.
7. Select the **Application Load Balancer** and **Dedicated** options, if they aren't already selected.


Elastic Beanstalk > Applications > getting-started-app

Modify load balancer

Load balancer type

(●) Application Load Balancer
Application layer load balancer—routing HTTP and HTTPS traffic based on protocol, port, and route to environment processes.

( ) Classic Load Balancer
Previous generation — HTTP, HTTPS, and TCP

( ) Network Load Balancer
Ultra-high performance and static IP addresses for your application.

(●) Dedicated
Use a load balancer that Elastic Beanstalk creates exclusively for this environment.

( ) Shared
Use a load balancer that someone in your account created. It can be shared among multiple Elastic Beanstalk environments.

8. Make any Application Load Balancer configuration changes that your environment requires.
9. Choose **Save**, and then make any other configuration changes that your environment requires.
10. Choose **Create environment**.

To configure a running environment's Application Load Balancer in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Load balancer** configuration category, choose **Edit**.

> **Note**
>
> If the **Load balancer** configuration category doesn't have an **Edit** button, your environment doesn't have a load balancer. To learn how to set one up, see Changing environment type.

5. Make the Application Load Balancer configuration changes that your environment requires.

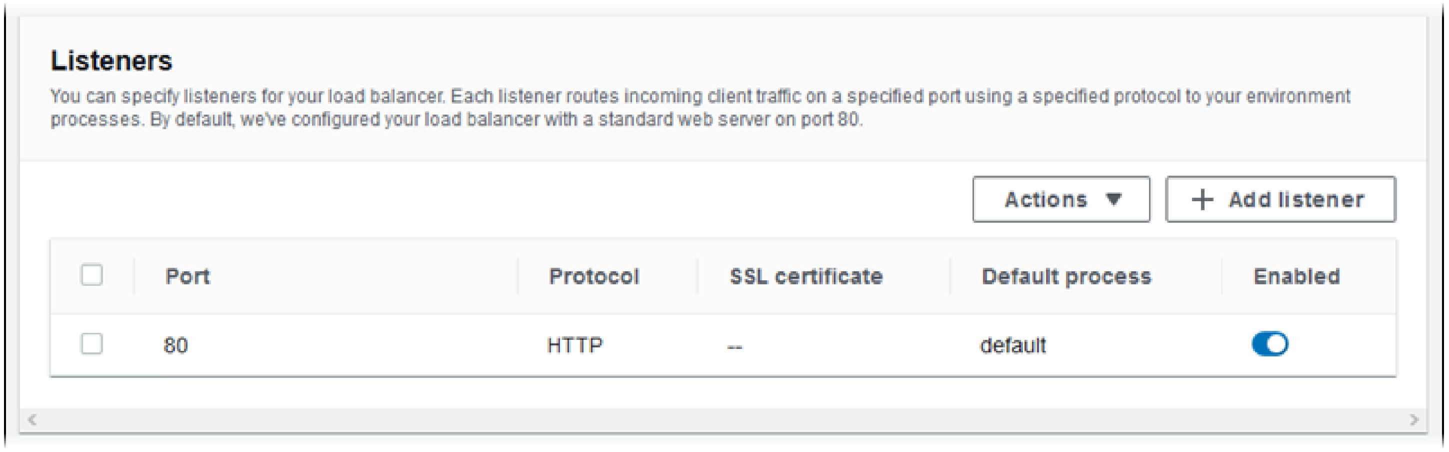
6. To save the changes choose **Apply** at the bottom of the page.

Application Load Balancer settings

- Listeners
- Processes
- Rules
- Access log capture

Listeners

Use this list to specify listeners for your load balancer. Each listener routes incoming client traffic on a specified port using a specified protocol to one or more processes on your instances. Initially, the list shows the default listener, which routes incoming HTTP traffic on port 80 to a process named **default**.



Listeners

You can specify listeners for your load balancer. Each listener routes incoming client traffic on a specified port using a specified protocol to your environment processes. By default, we've configured your load balancer with a standard web server on port 80.

	Port	Protocol	SSL certificate	Default process	Enabled
☐	80	HTTP	--	default	●

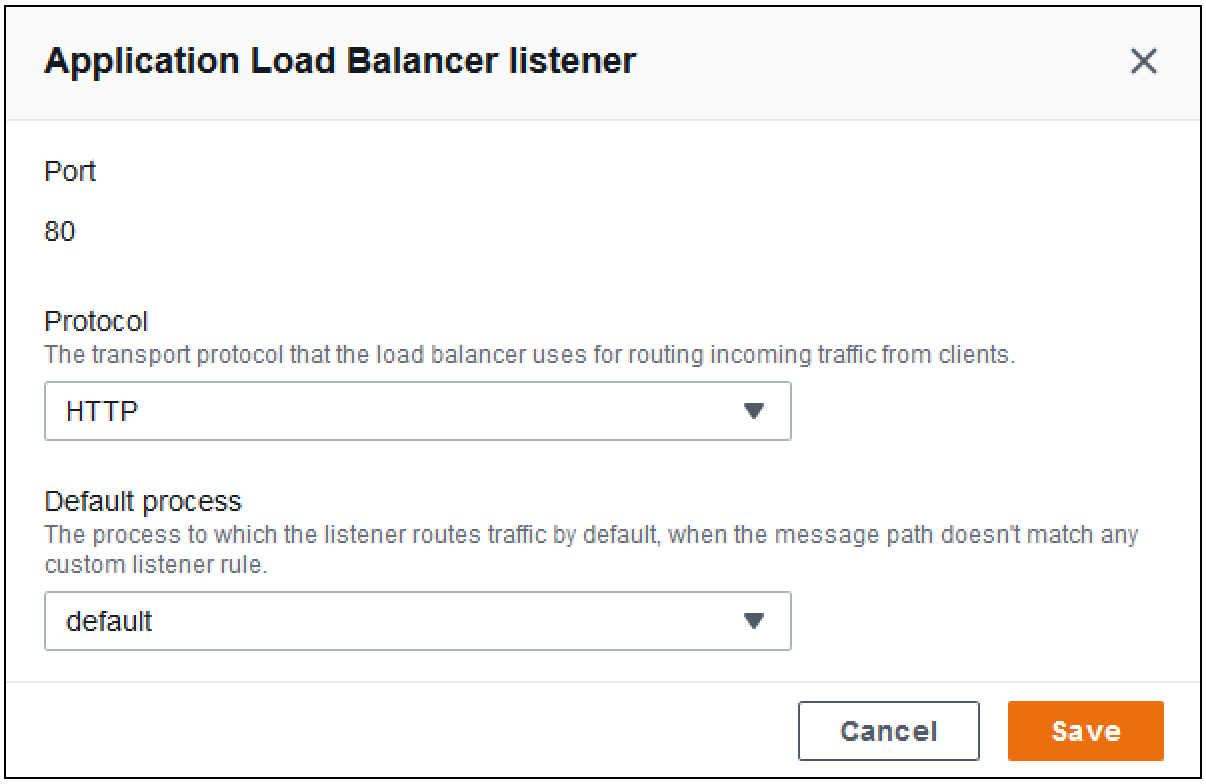
To configure an existing listener

1. Select the check box next to its table entry, and then choose **Actions**, **Edit**.
2. Use the **Application Load Balancer listener** dialog box to edit settings, and then choose **Save**.

To add a listener

1. Choose **Add listener**.
2. In the **Application Load Balancer listener** dialog box, configure the settings you want, and then choose **Add**.

Use the **Application Load Balancer listener** dialog box settings to choose the port and protocol on which the listener listens to traffic, and the process to route the traffic to. If you choose the HTTPS protocol, configure SSL settings.



Application Load Balancer listener

Port

80

Protocol
The transport protocol that the load balancer uses for routing incoming traffic from clients.

HTTP

Default process
The process to which the listener routes traffic by default, when the message path doesn't match any custom listener rule.

default

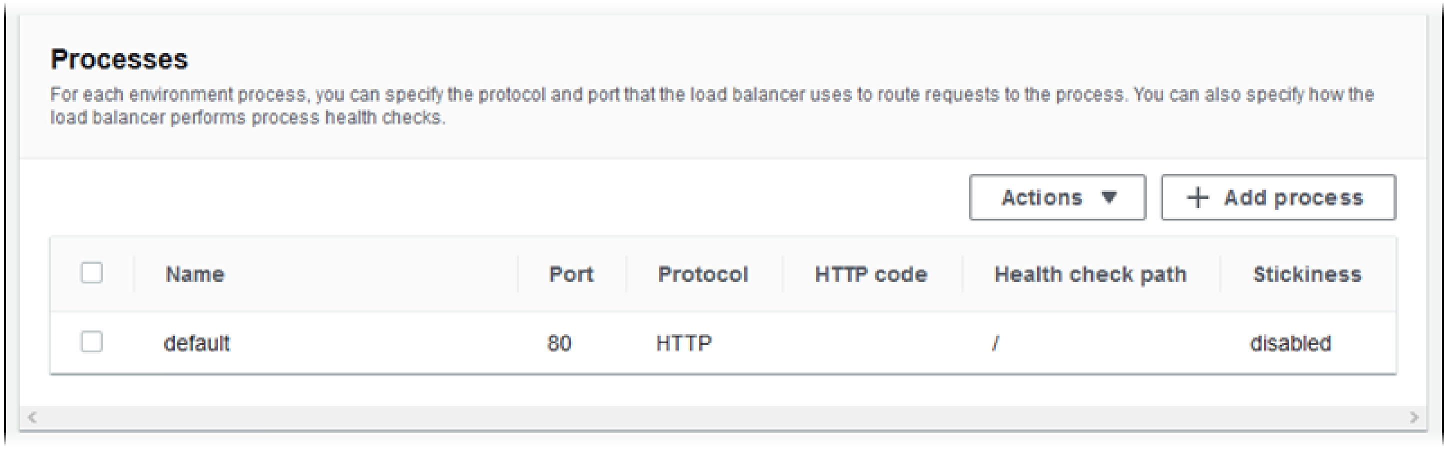
Before you can configure an HTTPS listener, ensure that you have a valid SSL certificate. Do one of the following:

- If AWS Certificate Manager (ACM) is available in your AWS Region, create or import a certificate using ACM. For more information about requesting an ACM certificate, see Request a Certificate in the *AWS Certificate Manager User Guide*. For more information about importing third-party certificates into ACM, see Importing Certificates in the *AWS Certificate Manager User Guide*.
- If ACM isn't available in your AWS Region, upload your existing certificate and key to IAM. For more information about creating and uploading certificates to IAM, see Working with Server Certificates in the *IAM User Guide*.

For more detail on configuring HTTPS and working with certificates in Elastic Beanstalk, see Configuring HTTPS for your Elastic Beanstalk environment.

Processes

Use this list to specify processes for your load balancer. A process is a target for listeners to route traffic to. Each listener routes incoming client traffic on a specified port using a specified protocol to one or more processes on your instances. Initially, the list shows the default process, which listens to incoming HTTP traffic on port 80.



Processes

For each environment process, you can specify the protocol and port that the load balancer uses to route requests to the process. You can also specify how the load balancer performs process health checks.

	Name	Port	Protocol	HTTP code	Health check path	Stickiness
☐	default	80	HTTP		/	disabled

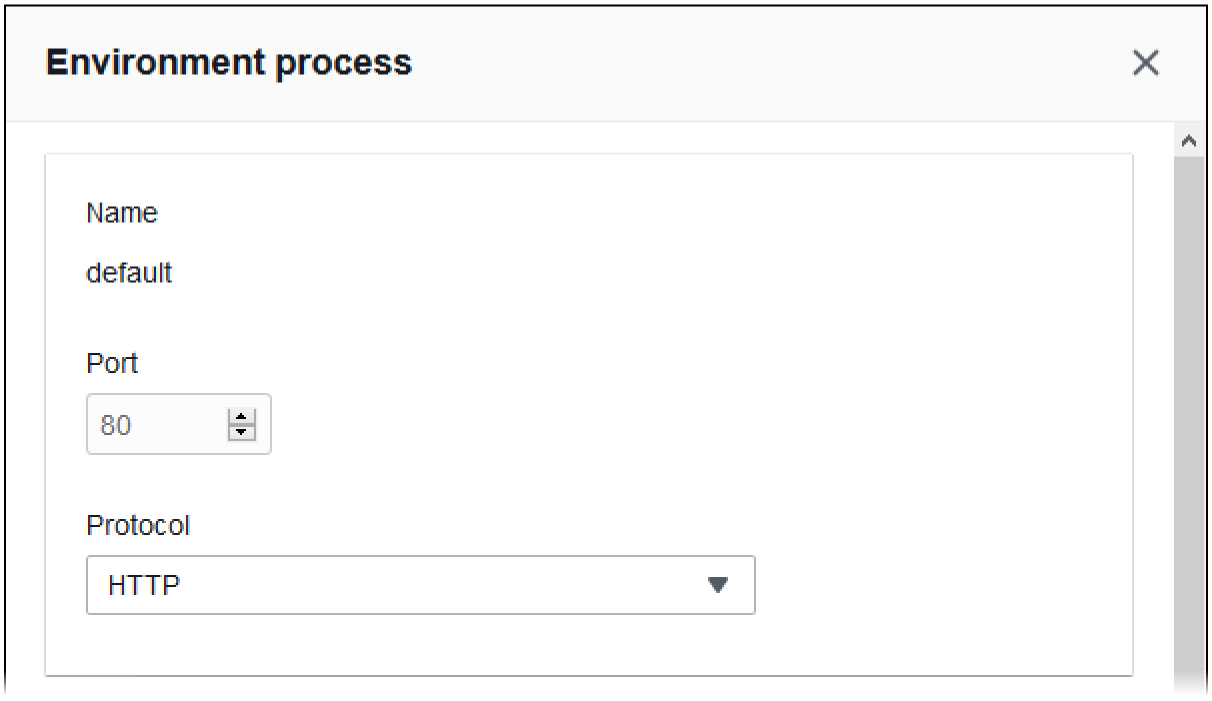
You can edit the settings of an existing process, or add a new process. To start editing a process on the list or adding a process to it, use the same steps listed for the listener list. The **Environment process** dialog box opens.

Application Load Balancer's environment process dialog box settings

- Definition
- Health check
- Sessions

Definition

Use these settings to define the process: its **Name**, and the **Port** and **Protocol** on which it listens to requests.


Environment process

Name

default

Port

80

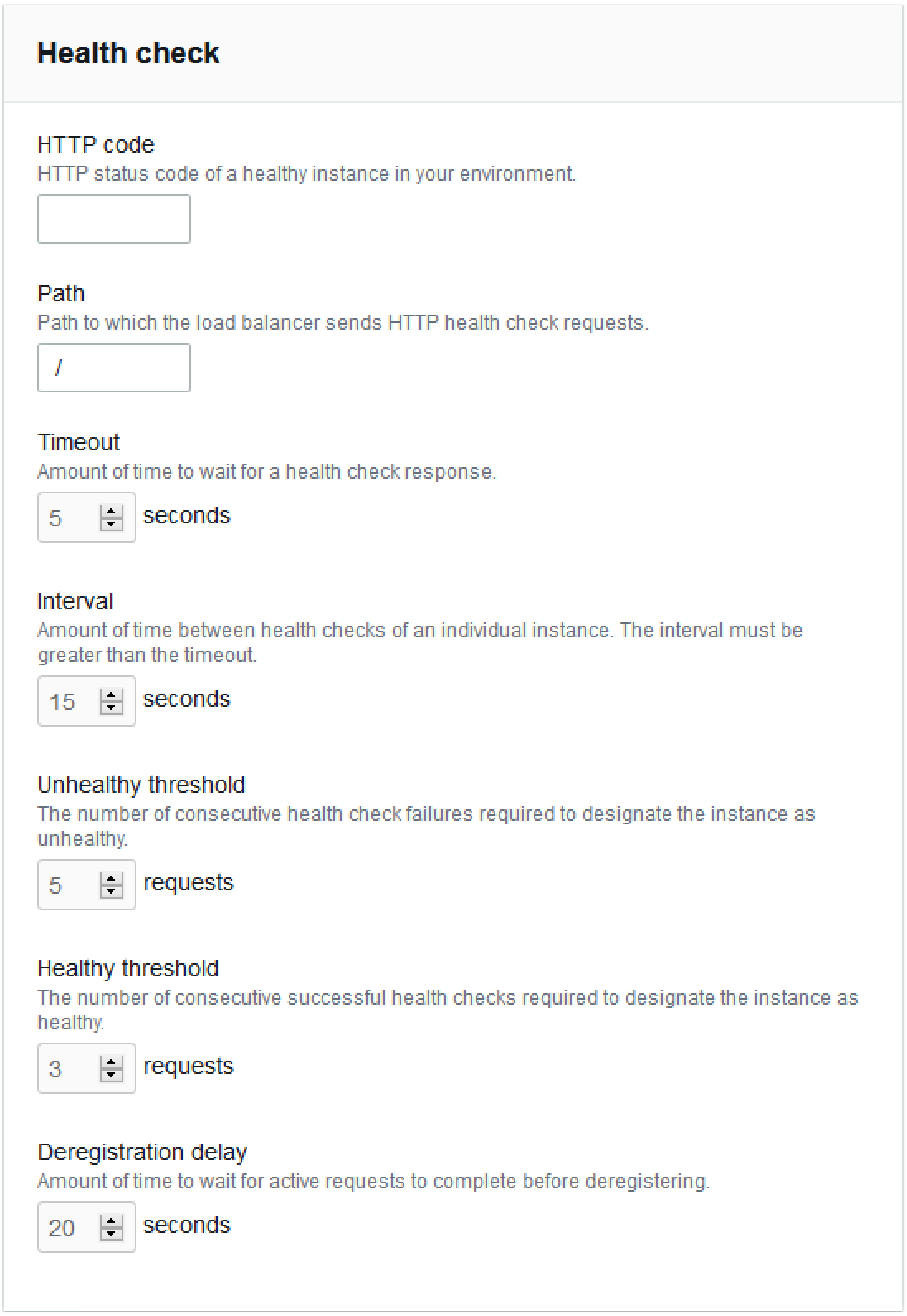
Protocol

HTTP

Health check

Use the following settings to configure process health checks:

- **HTTP code –** The HTTP status code designating a healthy process.
- **Path –** The health check request path for the process.
- **Timeout –** The amount of time, in seconds, to wait for a health check response.
- **Interval –** The amount of time, in seconds, between health checks of an individual instance. The interval must be greater than the timeout.
- **Unhealthy threshold**, **Healthy threshold –** The number of health checks that must fail or pass, respectively, before Elastic Load Balancing changes an instance's health state.
- **Deregistration delay –** The amount of time, in seconds, to wait for active requests to complete before deregistering an instance.



Health check

HTTP code
HTTP status code of a healthy instance in your environment.

Path
Path to which the load balancer sends HTTP health check requests.
/

Timeout
Amount of time to wait for a health check response.
5 seconds

Interval
Amount of time between health checks of an individual instance. The interval must be greater than the timeout.
15 seconds

Unhealthy threshold
The number of consecutive health check failures required to designate the instance as unhealthy.
5 requests

Healthy threshold
The number of consecutive successful health checks required to designate the instance as healthy.
3 requests

Deregistration delay
Amount of time to wait for active requests to complete before deregistering.
20 seconds

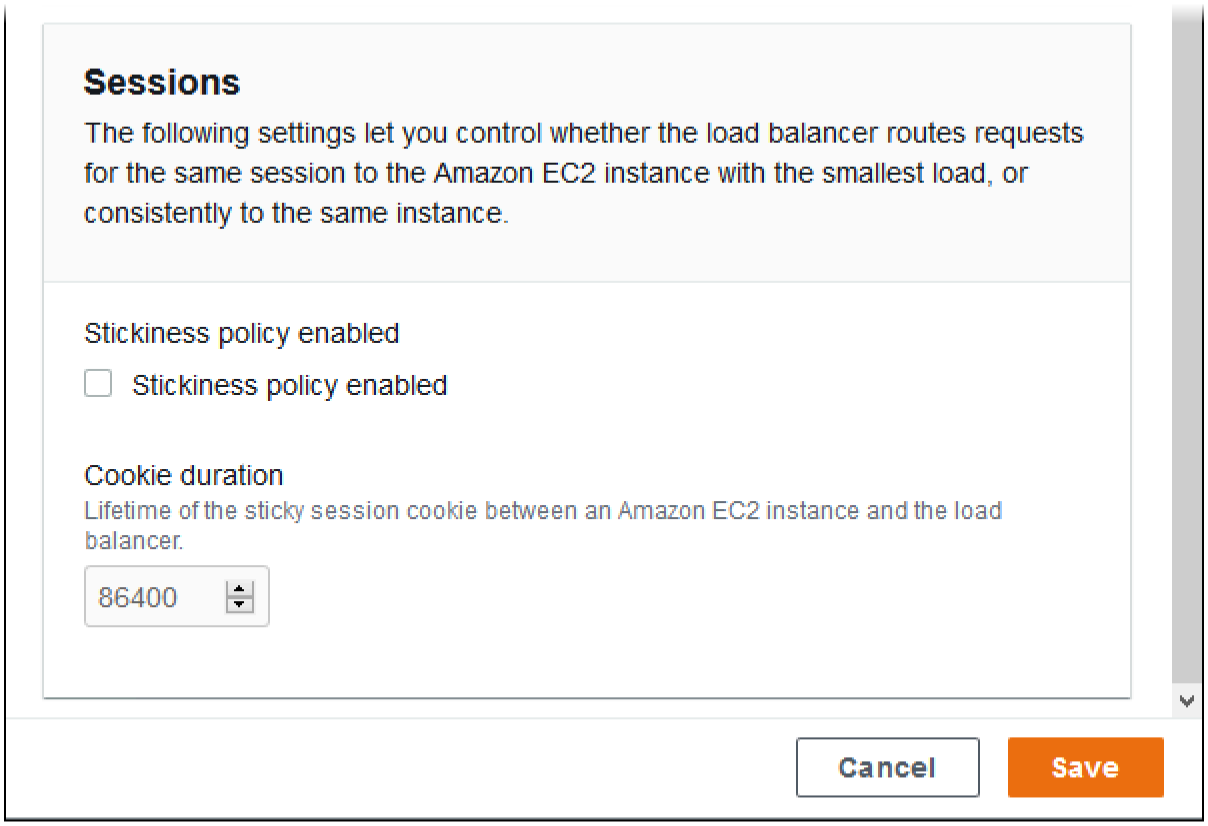
Note

The Elastic Load Balancing health check doesn't affect the health check behavior of an environment's Auto Scaling group. Instances that fail an Elastic Load Balancing health check are not automatically replaced by Amazon EC2 Auto Scaling unless you manually configure Amazon EC2 Auto Scaling to do so. See Auto Scaling health check setting for your Elastic Beanstalk environment for details.

For more information about health checks and how they influence your environment's overall health, see Basic health reporting.

Sessions

Select or clear the **Stickiness policy enabled** box to enable or disable sticky sessions. Use **Cookie duration** to configure a sticky session's duration, up to **`604800`** seconds.



Sessions

The following settings let you control whether the load balancer routes requests for the same session to the Amazon EC2 instance with the smallest load, or consistently to the same instance.

Stickiness policy enabled

☐ Stickiness policy enabled

Cookie duration

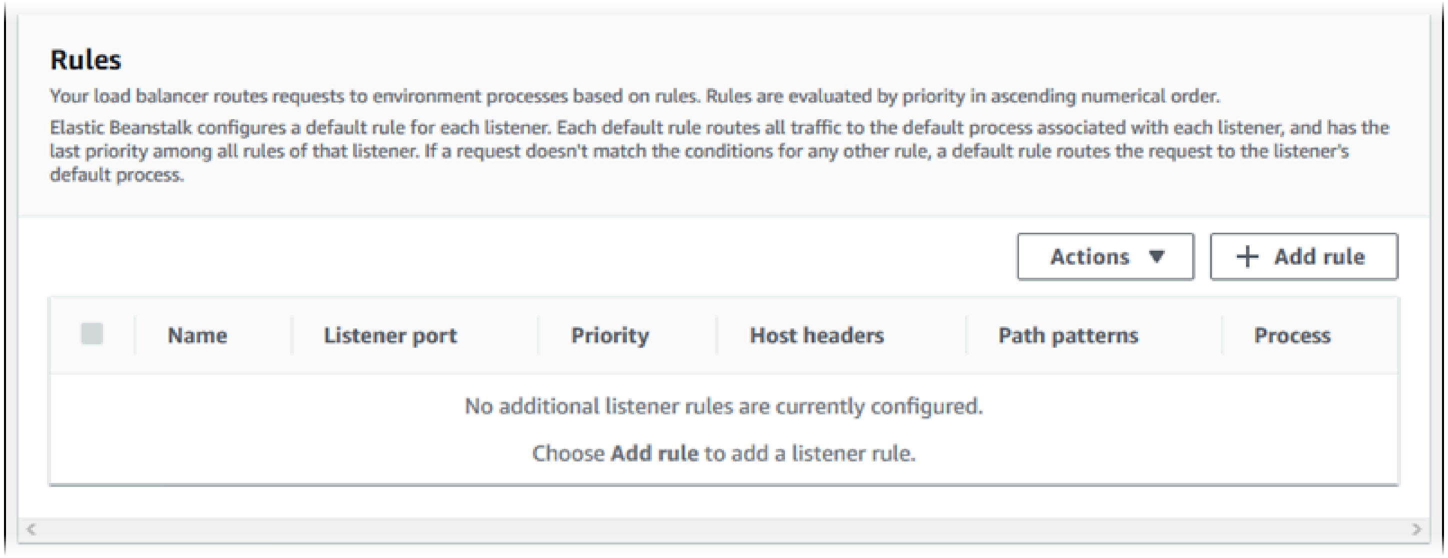
Lifetime of the sticky session cookie between an Amazon EC2 instance and the load balancer.

86400

Rules

Use this list to specify custom listener rules for your load balancer. A rule maps requests that the listener receives on a specific path pattern to a target process. Each listener can have multiple rules, routing requests on different paths to different processes on your instances.

Rules have numeric priorities that determine the precedence in which they are applied to incoming requests. For each new listener you add, Elastic Beanstalk adds a default rule that routes all the listener's traffic to the default process. The default rule's precedence is the lowest; it's applied if no other rule for the same listener matches the incoming request. Initially, if you haven't added custom rules, the list is empty. Default rules of all listeners aren't displayed.



Rules

Your load balancer routes requests to environment processes based on rules. Rules are evaluated by priority in ascending numerical order.

Elastic Beanstalk configures a default rule for each listener. Each default rule routes all traffic to the default process associated with each listener, and has the last priority among all rules of that listener. If a request doesn't match the conditions for any other rule, a default rule routes the request to the listener's default process.

	Name	Listener port	Priority	Host headers	Path patterns	Process

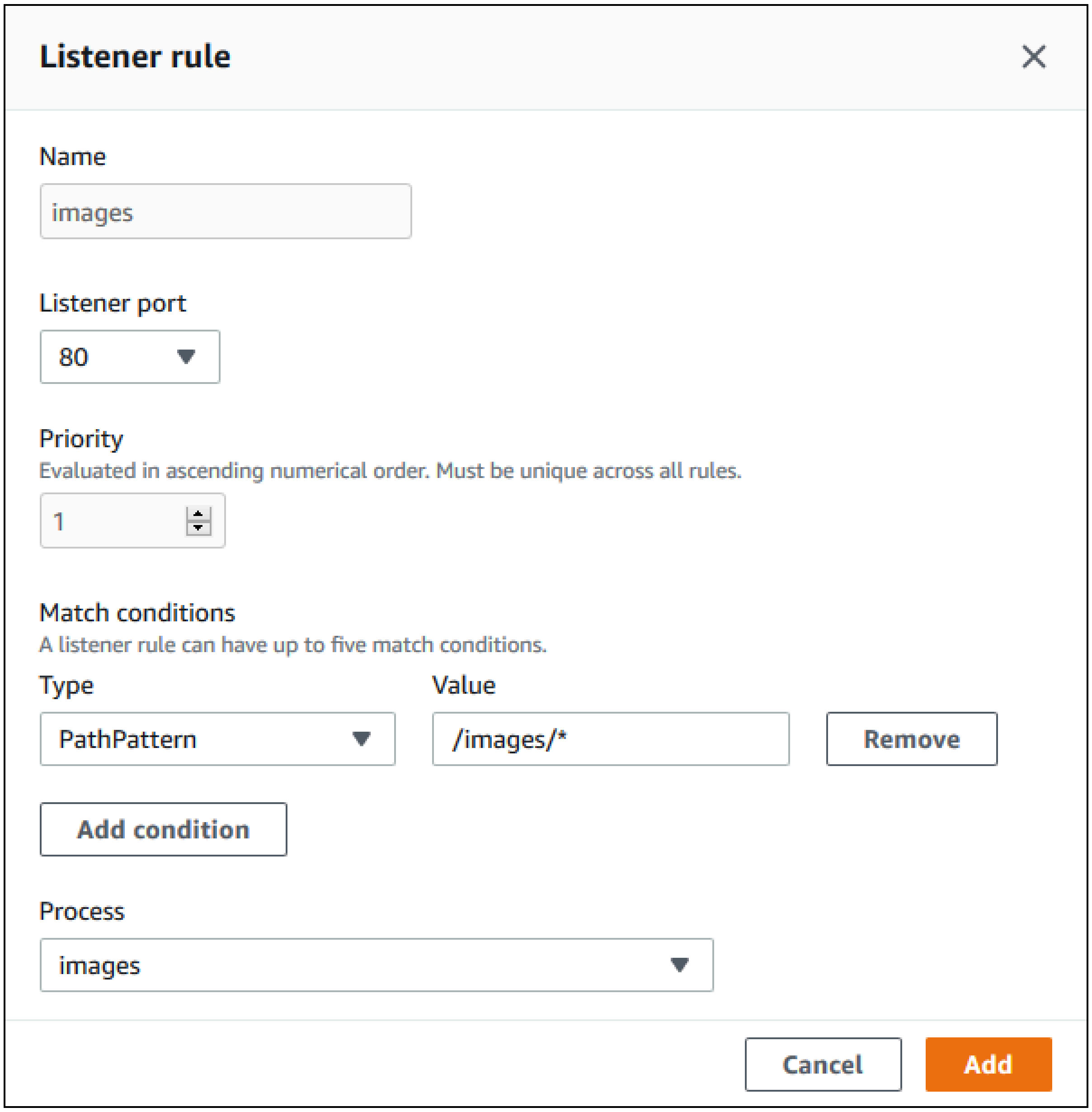
No additional listener rules are currently configured.

Choose **Add rule** to add a listener rule.

You can edit the settings of an existing rule, or add a new rule. To start editing a rule on the list or adding a rule to it, use the same steps listed for the listener list. The **Listener rule** dialog box opens, with the following settings:

- **Name** – The rule's name.
- **Listener port** – The port of the listener that the rule applies to.
- **Priority** – The rule's priority. A lower priority number has higher precedence. Priorities of a listener's rules must be unique.
- **Match conditions** – A list of request URL conditions that the rule applies to. There are two types of conditions: **HostHeader** (the URL's domain part), and **PathPattern** (the URL's path part). You can add up to five conditions. Each condition value is up to 128 characters long, and can include wildcard characters.
- **Process** – The process to which the load balancer routes requests that match the rule.

When editing any existing rule, you can't change its **Name** and **Listener port**.



Listener rule

Name: images

Listener port: 80

Priority
Evaluated in ascending numerical order. Must be unique across all rules.
1

Match conditions
A listener rule can have up to five match conditions.

Type	Value
PathPattern	/images/*

Process: images

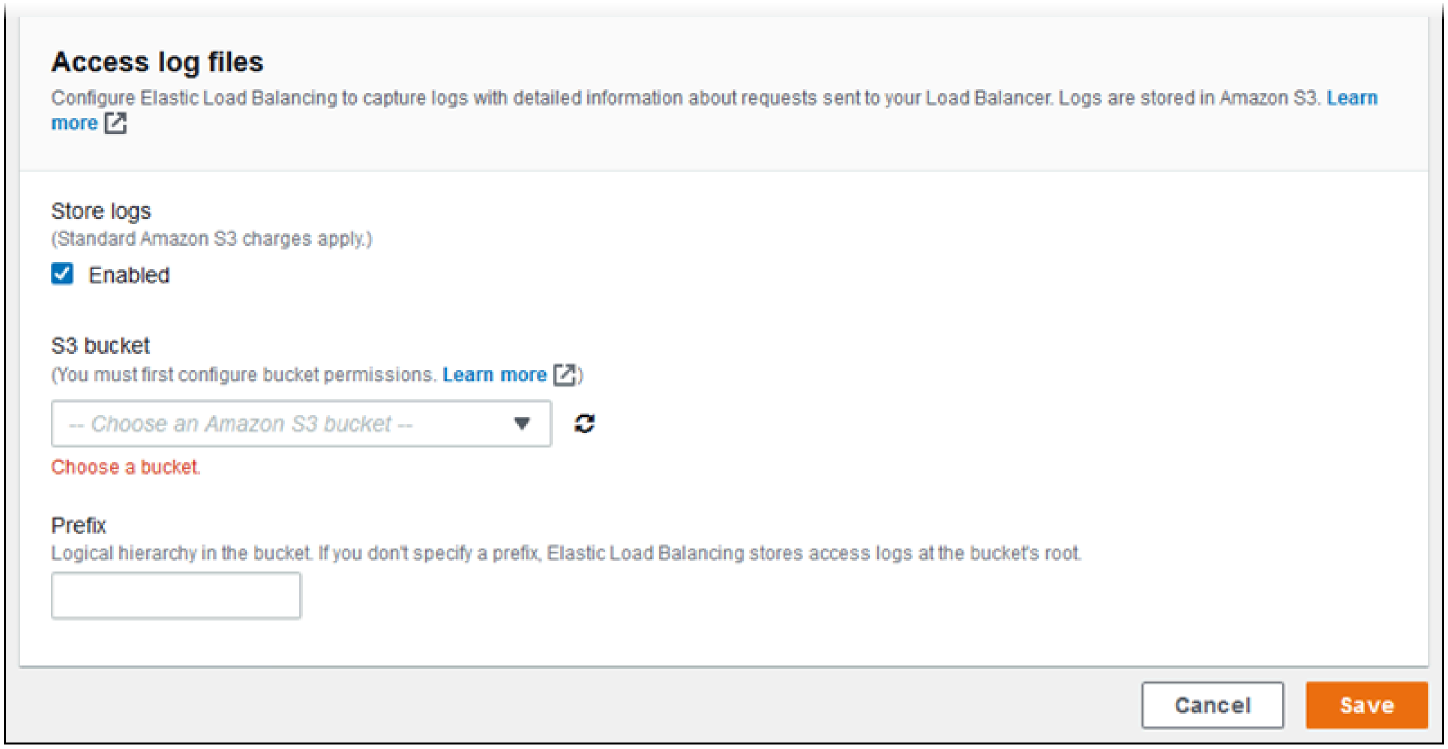
Access log capture

Use these settings to configure Elastic Load Balancing to capture logs with detailed information about requests sent to your Application Load Balancer. Access log capture is disabled by default. When **Store logs** is enabled, Elastic Load Balancing stores the logs in the **S3 bucket** that you configure. The **Prefix** setting specifies a top-level folder in the bucket for the logs. Elastic Load Balancing places the logs in a folder named `AWSLogs` under your prefix. If you don't specify a prefix, Elastic Load Balancing places its folder at the root level of the bucket.

Note

If the Amazon S3 bucket that you configure for access log capture isn't the bucket that Elastic Beanstalk created for your account, be sure to add a user policy with the appropriate permissions to your AWS Identity and Access Management (IAM) users. The managed user policies that Elastic Beanstalk provides only cover permissions to Elastic Beanstalk-managed resources.

For details about access logs, including permissions and other requirements, see Access logs for your Application Load Balancer.



Access log files

Configure Elastic Load Balancing to capture logs with detailed information about requests sent to your Load Balancer. Logs are stored in Amazon S3. Learn more

Store logs
(Standard Amazon S3 charges apply.)
☑ Enabled

S3 bucket
(You must first configure bucket permissions. Learn more)
-- Choose an Amazon S3 bucket --
Choose a bucket.

Prefix
Logical hierarchy in the bucket. If you don't specify a prefix, Elastic Load Balancing stores access logs at the bucket's root.

Example: Application Load Balancer with a secure listener and two processes

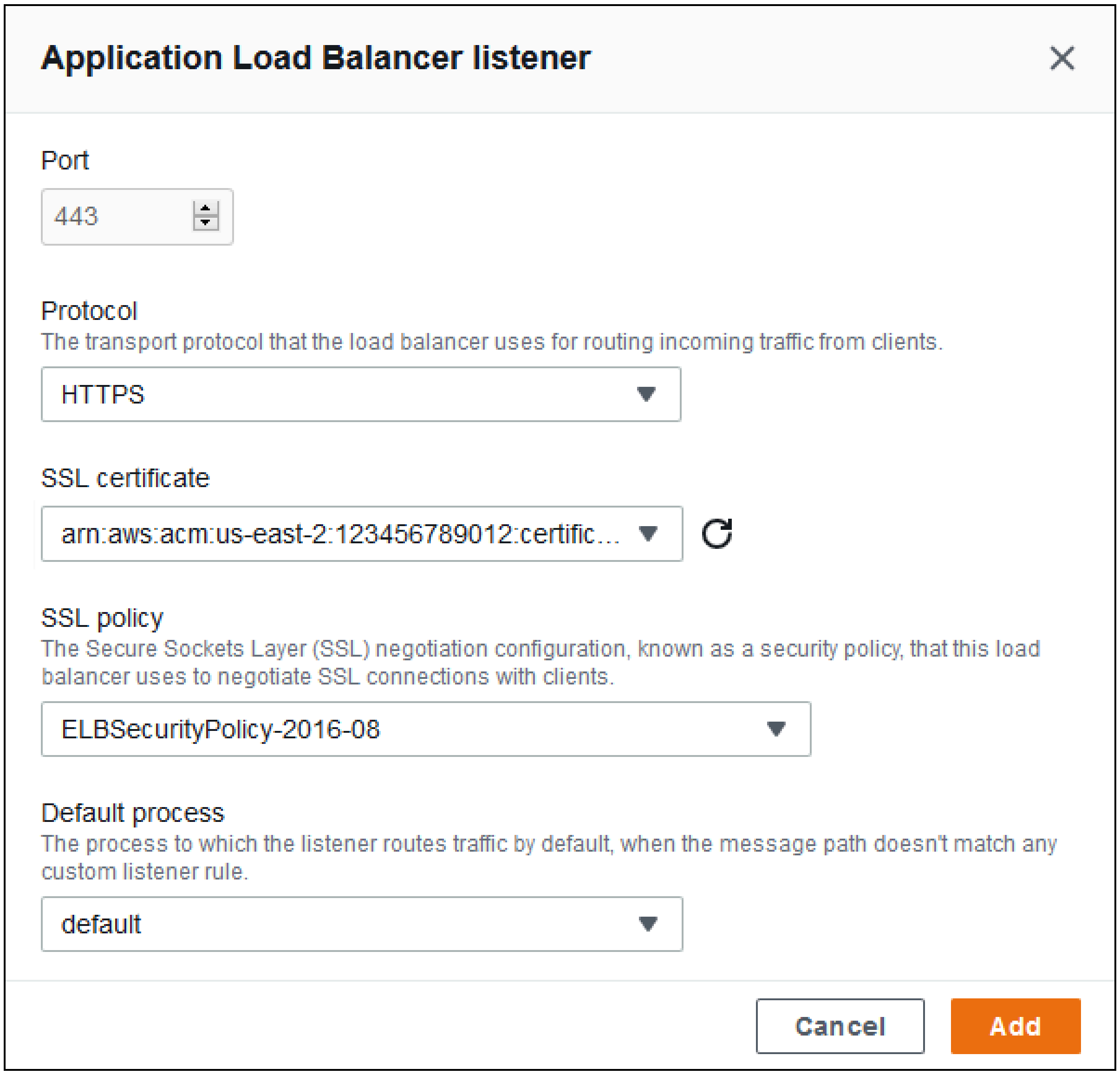
In this example, your application requires end-to-end traffic encryption and a separate process for handling administrative requests.

To configure your environment's Application Load Balancer to meet these requirements, you remove the default listener, add an HTTPS listener, indicate that the default process listens to port 443 on HTTPS, and add a process and a listener rule for admin traffic on a different path.

To configure the load balancer for this example

1. *Add a secure listener.* For **Port**, type **443**. For **Protocol**, select **HTTPS**. For **SSL certificate**, select the ARN of your SSL certificate. For example, **`arn:aws:iam::123456789012:server-certificate/abc/certs/build`**, or **`arn:aws:acm:us-east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678`**.

   For **Default process**, keep **`default`** selected.

   Application Load Balancer listener

   Port

   443

   Protocol

   The transport protocol that the load balancer uses for routing incoming traffic from clients.

   HTTPS

   SSL certificate

   arn:aws:acm:us-east-2:123456789012:certific...

   SSL policy

   The Secure Sockets Layer (SSL) negotiation configuration, known as a security policy, that this load balancer uses to negotiate SSL connections with clients.

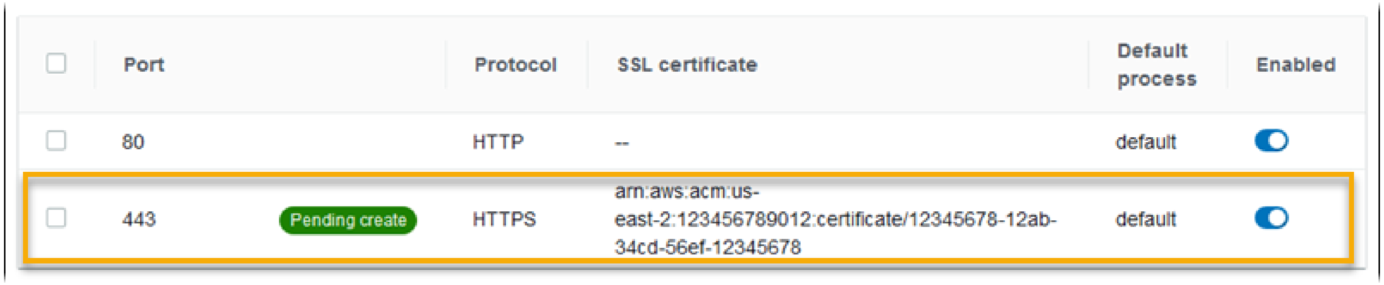
   ELBSecurityPolicy-2016-08

   Default process

   The process to which the listener routes traffic by default, when the message path doesn't match any custom listener rule.

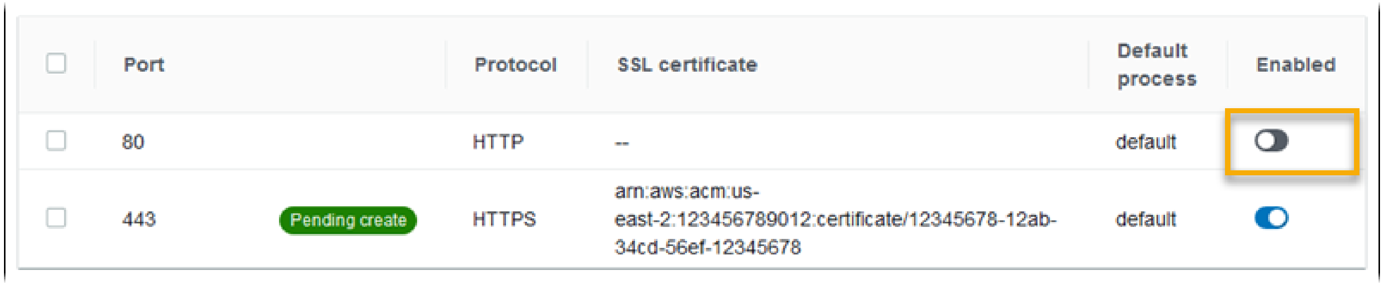
   default

   You can now see your additional listener on the list.



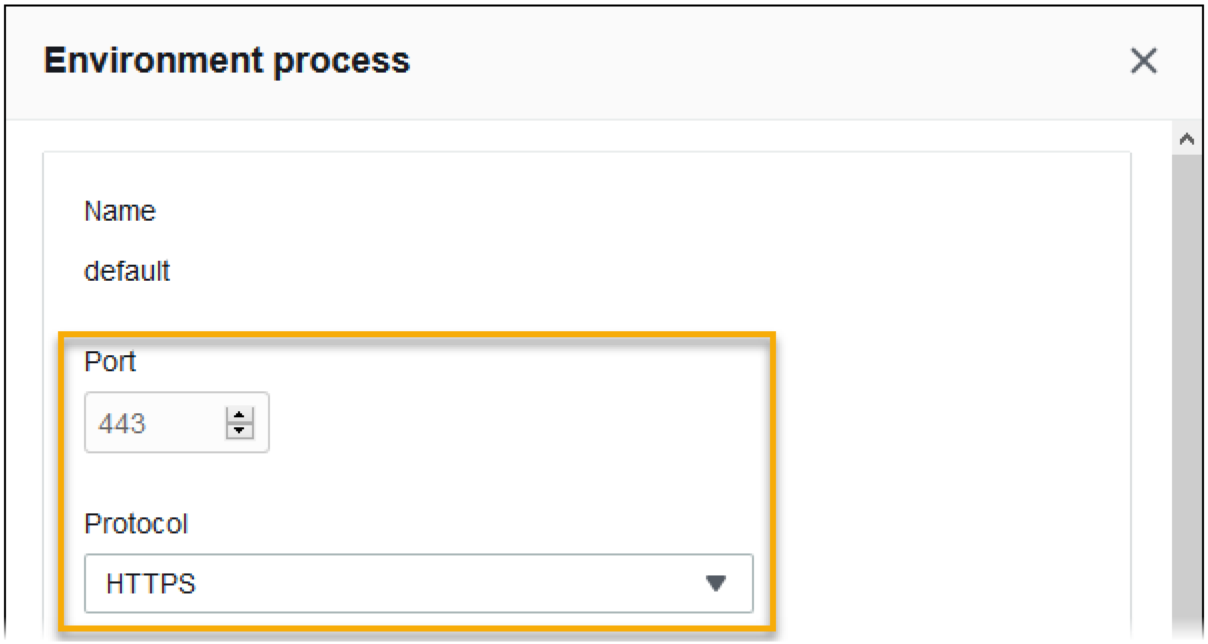
	Port		Protocol	SSL certificate	Default process	Enabled
☐	80		HTTP	--	default	●
☐	443	Pending create	HTTPS	arn:aws:acm:us-east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678	default	●

2. *Disable the default port 80 HTTP listener.* For the default listener, turn off the **Enabled** option.



	Port		Protocol	SSL certificate	Default process	Enabled
☐	80		HTTP	--	default	○
☐	443	Pending create	HTTPS	arn:aws:acm:us-east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678	default	●

3. *Configure the default process to HTTPS.* Select the default process, and then for **Actions**, choose **Edit**. For **Port**, type **443**. For **Protocol**, select **HTTPS**.



Environment process

Name

default

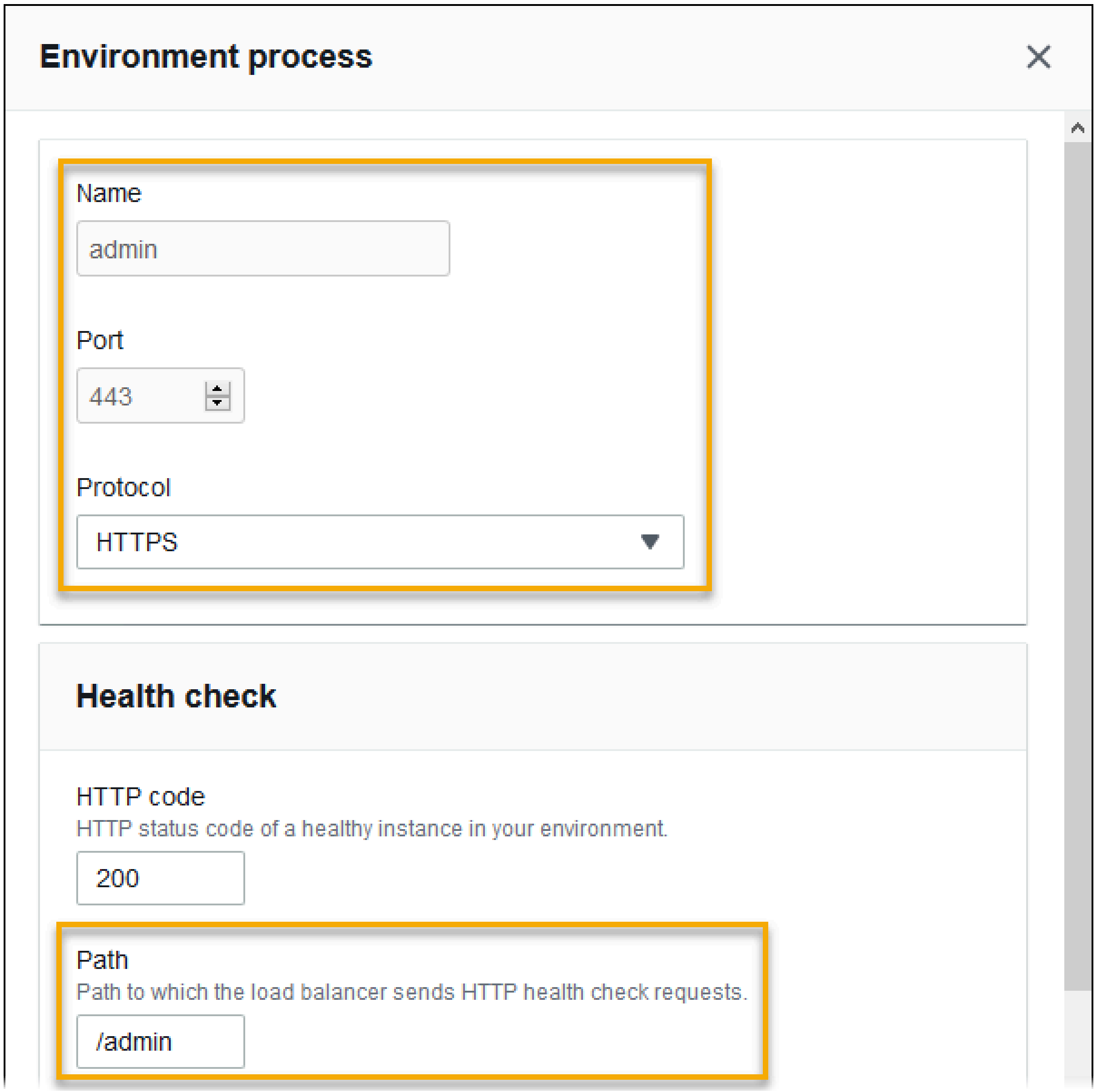
Port

443

Protocol

HTTPS

4. *Add an admin process.* For **Name**, type **admin**. For **Port**, type **443**. For **Protocol**, select **HTTPS**. Under **Health check**, for **Path** type **/admin**.



Environment process

Name

admin

Port

443

Protocol

HTTPS

Health check

HTTP code

HTTP status code of a healthy instance in your environment.

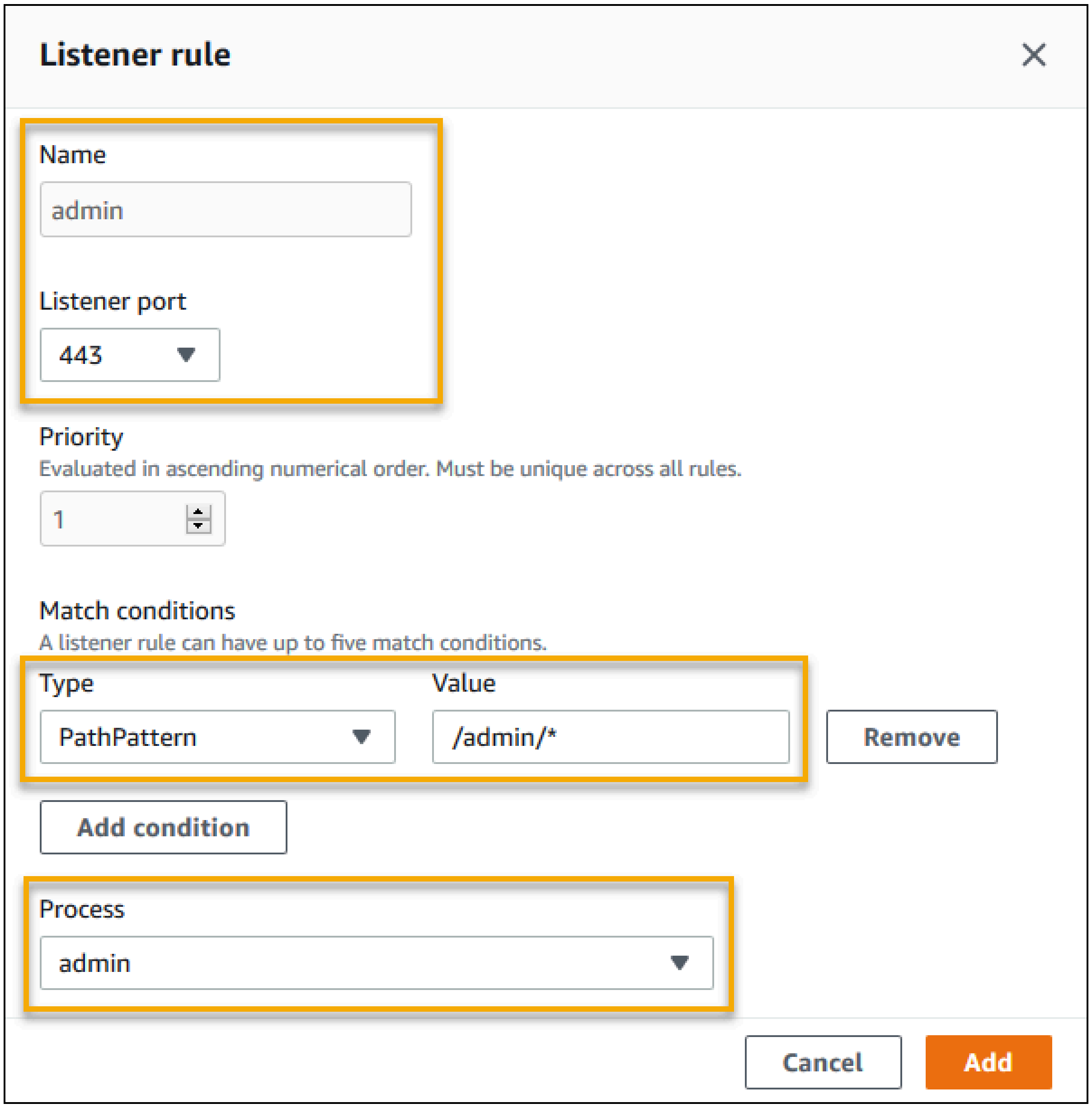
200

Path

Path to which the load balancer sends HTTP health check requests.

/admin

5. *Add a rule for admin traffic.* For **Name**, type **`admin`**. For **Listener port**, type **`443`**. For **Match conditions**, add a **PathPattern** with the value **`/admin/*`**. For **Process**, select **`admin`**.



Configuring an Application Load Balancer using the EB CLI

The EB CLI prompts you to choose a load balancer type when you run eb create.

```
$ eb create
Enter Environment Name
(default is my-app): test-env
Enter DNS CNAME prefix
(default is my-app): test-env-DLW24ED23SF

Select a load balancer type
```

```
1) classic
2) application
3) network
(default is 2):
```

You can also specify a load balancer type with the `--elb-type` option.

```
$ eb create test-env --elb-type application
```

Application Load Balancer namespaces

You can find settings related to Application Load Balancers in the following namespaces:

- `aws:elasticbeanstalk:environment` – Choose the load balancer type for the environment. The value for an Application Load Balancer is `application`.

  You can't set this option in configuration files (.Ebextensions).
- `aws:elbv2:loadbalancer` – Configure access logs and other settings that apply to the Application Load Balancer as a whole.
- `aws:elbv2:listener` – Configure listeners on the Application Load Balancer. These settings map to the settings in `aws:elb:listener` for Classic Load Balancers.
- `aws:elbv2:listenerrule` – Configure rules that route traffic to different processes, depending on the request path. Rules are unique to Application Load Balancers.
- `aws:elasticbeanstalk:environment:process` – Configure health checks and specify the port and protocol for the processes that run on your environment's instances. The port and protocol settings map to the instance port and instance protocol settings in `aws:elb:listener` for a listener on a Classic Load Balancer. Health check settings map to the settings in the `aws:elb:healthcheck` and `aws:elasticbeanstalk:application` namespaces.

Example .ebextensions/alb-access-logs.config

The following configuration file enables access log uploads for an environment with an Application Load Balancer.

```
option_settings:
  aws:elbv2:loadbalancer:
    AccessLogsS3Bucket: amzn-s3-demo-bucket
    AccessLogsS3Enabled: 'true'
    AccessLogsS3Prefix: beanstalk-alb
```

Example .ebextensions/alb-default-process.config

The following configuration file modifies health check and stickiness settings on the default process.

```
option_settings:
  aws:elasticbeanstalk:environment:process:default:
    DeregistrationDelay: '20'
    HealthCheckInterval: '15'
    HealthCheckPath: /
    HealthCheckTimeout: '5'
    HealthyThresholdCount: '3'
    UnhealthyThresholdCount: '5'
    Port: '80'
    Protocol: HTTP
    StickinessEnabled: 'true'
    StickinessLBCookieDuration: '43200'
```

Example .ebextensions/alb-secure-listener.config

The following configuration file adds a secure listener and a matching process on port 443.

```
option_settings:
  aws:elbv2:listener:443:
    DefaultProcess: https
    ListenerEnabled: 'true'
    Protocol: HTTPS
    SSLCertificateArns: arn:aws:acm:us-
east-2:123456789012:certificate/21324896-0fa4-412b-bf6f-f362d6eb6dd7
  aws:elasticbeanstalk:environment:process:https:
    Port: '443'
    Protocol: HTTPS
```

Example .ebextensions/alb-admin-rule.config

The following configuration file adds a secure listener with a rule that routes traffic with a request path of `/admin` to a process named `admin` that listens on port 4443.

```
option_settings:
  aws:elbv2:listener:443:
    DefaultProcess: https
    ListenerEnabled: 'true'
    Protocol: HTTPS
    Rules: admin
    SSLCertificateArns: arn:aws:acm:us-
east-2:123456789012:certificate/21324896-0fa4-412b-bf6f-f362d6eb6dd7
  aws:elasticbeanstalk:environment:process:https:
    Port: '443'
    Protocol: HTTPS
  aws:elasticbeanstalk:environment:process:admin:
    HealthCheckPath: /admin
    Port: '4443'
    Protocol: HTTPS
  aws:elbv2:listenerrule:admin:
    PathPatterns: /admin/*
    Priority: 1
    Process: admin
```

Configuring a shared Application Load Balancer

When you enable load balancing, your AWS Elastic Beanstalk environment is equipped with an Elastic Load Balancing load balancer to distribute traffic among the instances in your environment. Elastic Load Balancing supports several load balancer types. To learn about them, see the Elastic Load Balancing User Guide. Elastic Beanstalk can create a load balancer for you, or enable you to specify a shared load balancer that you've created.

This topic describes the configuration of a shared Application Load Balancer that you create and associate with your environment. See also the section called “Application Load Balancer”. For information about configuring all the load balancer types that Elastic Beanstalk supports, see Load balancer for your Elastic Beanstalk environment.

Note

You can choose the type of load balancer that your environment uses only during environment creation. You can change settings to manage the behavior of your running environment's load balancer, but you can't change its type. You also can't switch from a dedicated to a shared load balancer or vice versa.

Introduction

A *shared load balancer* is a load balancer that you create and manage yourself using the Amazon Elastic Compute Cloud (Amazon EC2) service, and then use in multiple Elastic Beanstalk environments.

When you create a load-balanced, scaling environment and choose to use an Application Load Balancer, Elastic Beanstalk creates a load balancer dedicated to your environment by default. To learn what an Application Load Balancer is and how it works in an Elastic Beanstalk environment, see the introduction to configuring an Application Load Balancer for Elastic Beanstalk.

In some situations you might want to save the cost of having multiple dedicated load balancers. This can be helpful when you have multiple environments, for example, if your application is a suite of microservices instead of a monolithic service. In such cases you can choose to use a shared load balancer.

To use a shared load balancer, first create it in Amazon EC2 and add one or more listeners. During the creation of an Elastic Beanstalk environment, you then provide the load balancer and choose a listener port. Elastic Beanstalk associates the listener with the default process in your environment. You can add custom listener rules to route traffic from specific host headers and paths to other environment processes.

Elastic Beanstalk adds a tag to the shared load balancer. The tag name is `elasticbeanstalk:shared-elb-environment-count`, and its value is the number of environments sharing this load balancer.

Using a shared load balancer is different from using a dedicated one in several ways.

Regarding	Dedicated Application Load Balancer	Shared Application Load Balancer
Manageme t	Elastic Beanstalk creates and manages the load balancer, listeners, listener rules, and processes (target groups). Elastic Beanstalk also removes them when you terminate your environment. Elastic Beanstalk	You create and manage the load balancer and listeners outside of Elastic Beanstalk. Elastic Beanstalk creates and manages a default rule and a default process, and you can add rules and processes. Elastic Beanstalk removes the listener rules and processes that were added during environment creation.

Regarding	Dedicated Application Load Balancer	Shared Application Load Balancer
	can set load balancer access log capture, if you choose that option.	
Listener rules	Elastic Beanstalk creates a default rule for each listener, to route all traffic to the listener's default process.	Elastic Beanstalk associates a default rule only with a port 80 listener, if one exists. If you choose a different default listener port, you have to associate the default rule with it (the Elastic Beanstalk console and EB CLI do this for you). To resolve listener rule condition conflicts across environments sharing the load balancer, Elastic Beanstalk adds the environme nt's CNAME to the listener rule as a host header condition. Elastic Beanstalk treats rule priority settings as relative across environments sharing the load balancer, and maps them to absolute priorities during creation.
Security groups	Elastic Beanstalk creates a default security group and attaches it to the load balancer.	You can configure one or more security groups to use for the load balancer. If you don't, Elastic Beanstalk checks if an existing security group that Elastic Beanstalk manages is already attached to the load balancer. If not, Elastic Beanstalk creates a security group and attaches it to the load balancer. Elastic Beanstalk deletes this security group when the last environment sharing the load balancer terminates.

Regarding	Dedicated Application Load Balancer	Shared Application Load Balancer
Updates	You can update your Application Load Balancer after environment creation. You can edit listeners, listener rules, and processes. You can configure load balancer access log capture.	You can't use Elastic Beanstalk to configure access log capture in your Application Load Balancer, and you can't update listeners and listener rules after environment creation. You can only update processes (target groups). To configure access log capture, and to update listeners and listener rules, use Amazon EC2.

Configuring a shared Application Load Balancer using the Elastic Beanstalk console

You can use the Elastic Beanstalk console to configure a shared Application Load Balancer during environment creation. You can select one of your account's sharable load balancers for use in the environment, select the default listener port, and configure additional processes and listener rules.

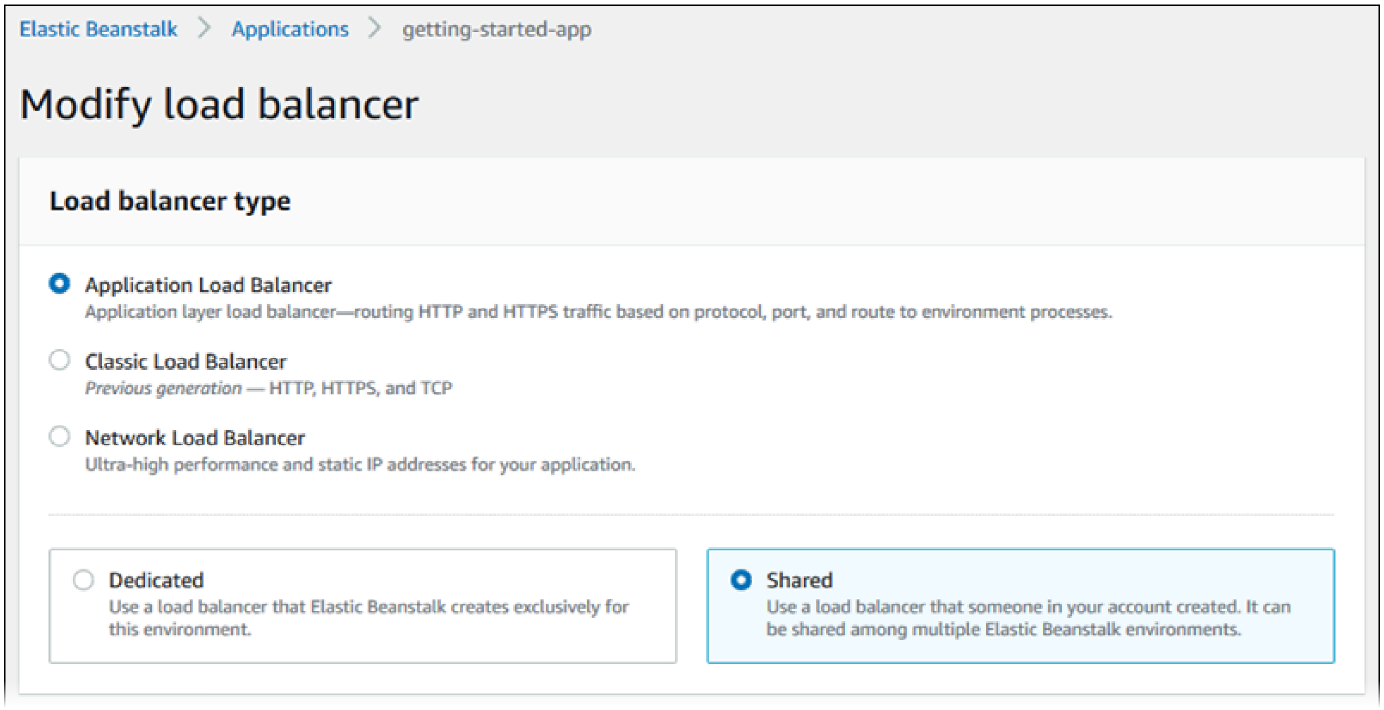
You can't edit your shared Application Load Balancer configuration in the Application Load Balancer console after your environment is created. To configure listeners, listener rules, processes (target groups), and access log capture, use Amazon EC2.

To configure an Application Load Balancer in the Elastic Beanstalk console during environment creation

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**.
3. Choose Create a new environment to start creating your environment.
4. On the wizard's main page, before choosing **Create environment**, choose **Configure more options**.
5. Choose the **High availability** configuration preset.

   Alternatively, in the **Capacity** configuration category, configure a **Load balanced** environment type. For details, see Capacity.
6. In the **Load balancer** configuration category, choose **Edit**.

7. Select the **Application Load Balancer** option, if it isn't already selected, and then select the **Shared** option.



Elastic Beanstalk > Applications > getting-started-app

Modify load balancer

Load balancer type

● Application Load Balancer
Application layer load balancer—routing HTTP and HTTPS traffic based on protocol, port, and route to environment processes.

○ Classic Load Balancer
Previous generation — HTTP, HTTPS, and TCP

○ Network Load Balancer
Ultra-high performance and static IP addresses for your application.

○ Dedicated
Use a load balancer that Elastic Beanstalk creates exclusively for this environment.

● Shared
Use a load balancer that someone in your account created. It can be shared among multiple Elastic Beanstalk environments.

8. Make any shared Application Load Balancer configuration changes that your environment requires.

9. Choose **Save**, and then make any other configuration changes that your environment requires.

10. Choose **Create environment**.

Shared Application Load Balancer settings

- Shared Application Load Balancer
- Processes
- Rules

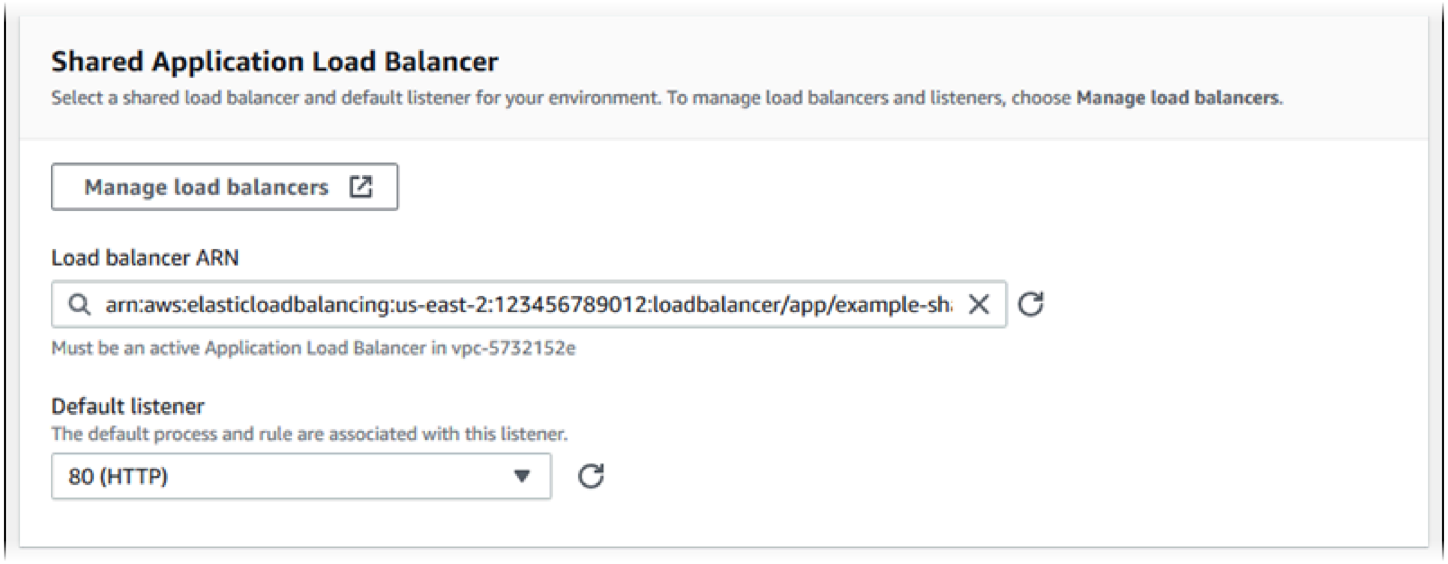
Shared Application Load Balancer

Use this section to choose a shared Application Load Balancer for your environment and configure default traffic routing.

Before you can configure a shared Application Load Balancer here, use Amazon EC2 to define at least one Application Load Balancer for sharing, with at least one listener, in your account. If you haven't done so already, you can choose **Manage load balancers**. Elastic Beanstalk opens the Amazon EC2 console in a new browser tab.

When you're done configuring shared load balancers outside of Elastic Beanstalk, configure the following settings on this console section:

- **Load balancer ARN** – The shared load balancer to use in this environment. Select from a list of load balancers or enter a load balancer Amazon Resource Name (ARN).
- **Default listener port** – A listener port that the shared load balancer listens on. Select from a list of existing listener ports. Traffic from this listener with the environment's CNAME in the host header is routed to a default process in this environment.



Shared Application Load Balancer

Select a shared load balancer and default listener for your environment. To manage load balancers and listeners, choose **Manage load balancers**.

Manage load balancers

Load balancer ARN

arn:aws:elasticloadbalancing:us-east-2:123456789012:loadbalancer/app/example-sh

Must be an active Application Load Balancer in vpc-5732152e

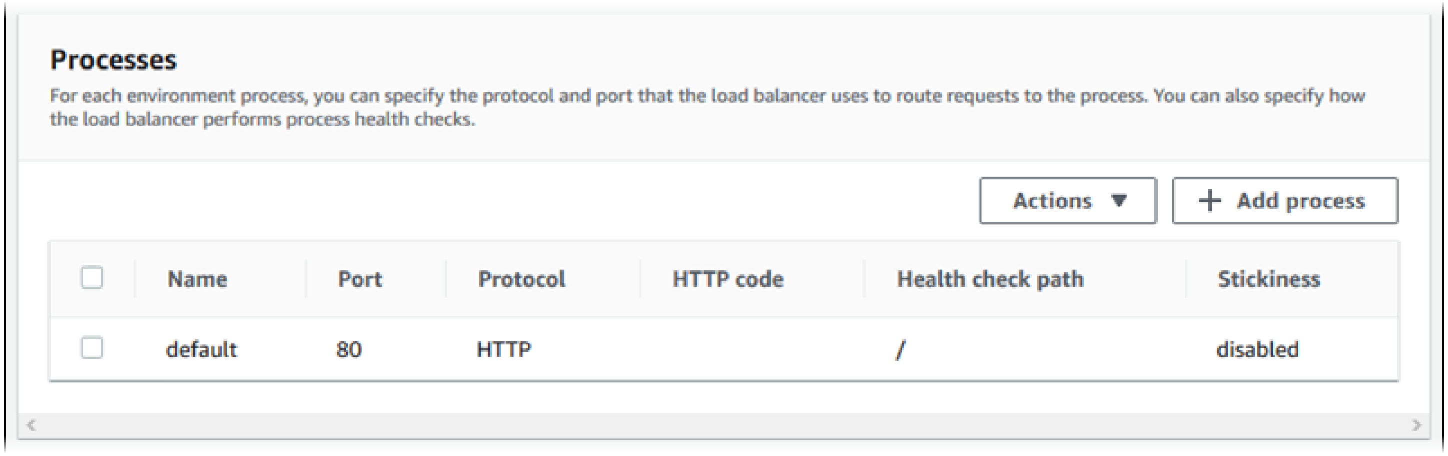
Default listener

The default process and rule are associated with this listener.

80 (HTTP)

Processes

Use this list to specify processes for your shared load balancer. A process is a target for listeners to route traffic to. Initially, the list shows the default process, which receives traffic from the default listener.



For each environment process, you can specify the protocol and port that the load balancer uses to route requests to the process. You can also specify how the load balancer performs process health checks.

Name	Port	Protocol	HTTP code	Health check path	Stickiness
default	80	HTTP		/	disabled

To configure an existing process

1. Select the check box next to its table entry, and then choose **Actions**, **Edit**.
2. Use the **Environment process** dialog box to edit settings, and then choose **Save**.

To add a process

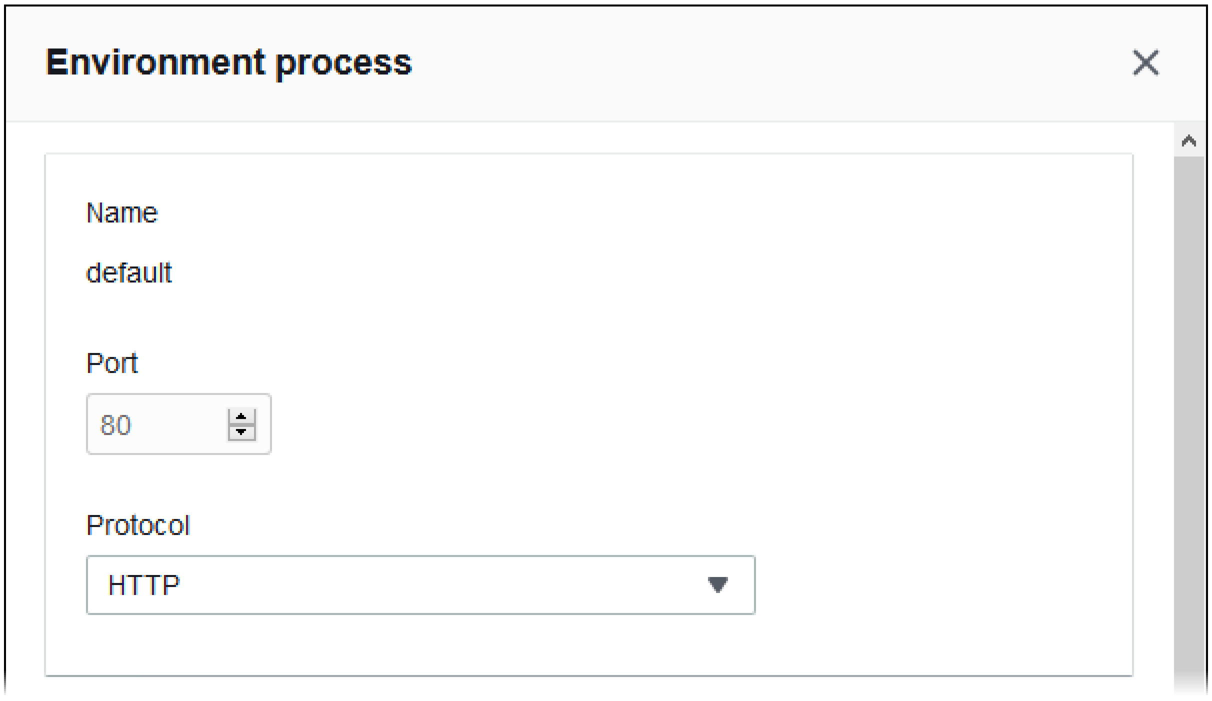
1. Choose **Add process**.
2. In the **Environment process** dialog box, configure the settings you want, and then choose **Add**.

Application Load Balancer's environment process dialog box settings

- Definition
- Health check
- Sessions

Definition

Use these settings to define the process: its **Name**, and the **Port** and **Protocol** on which it listens to requests.



Name

default

Port

80

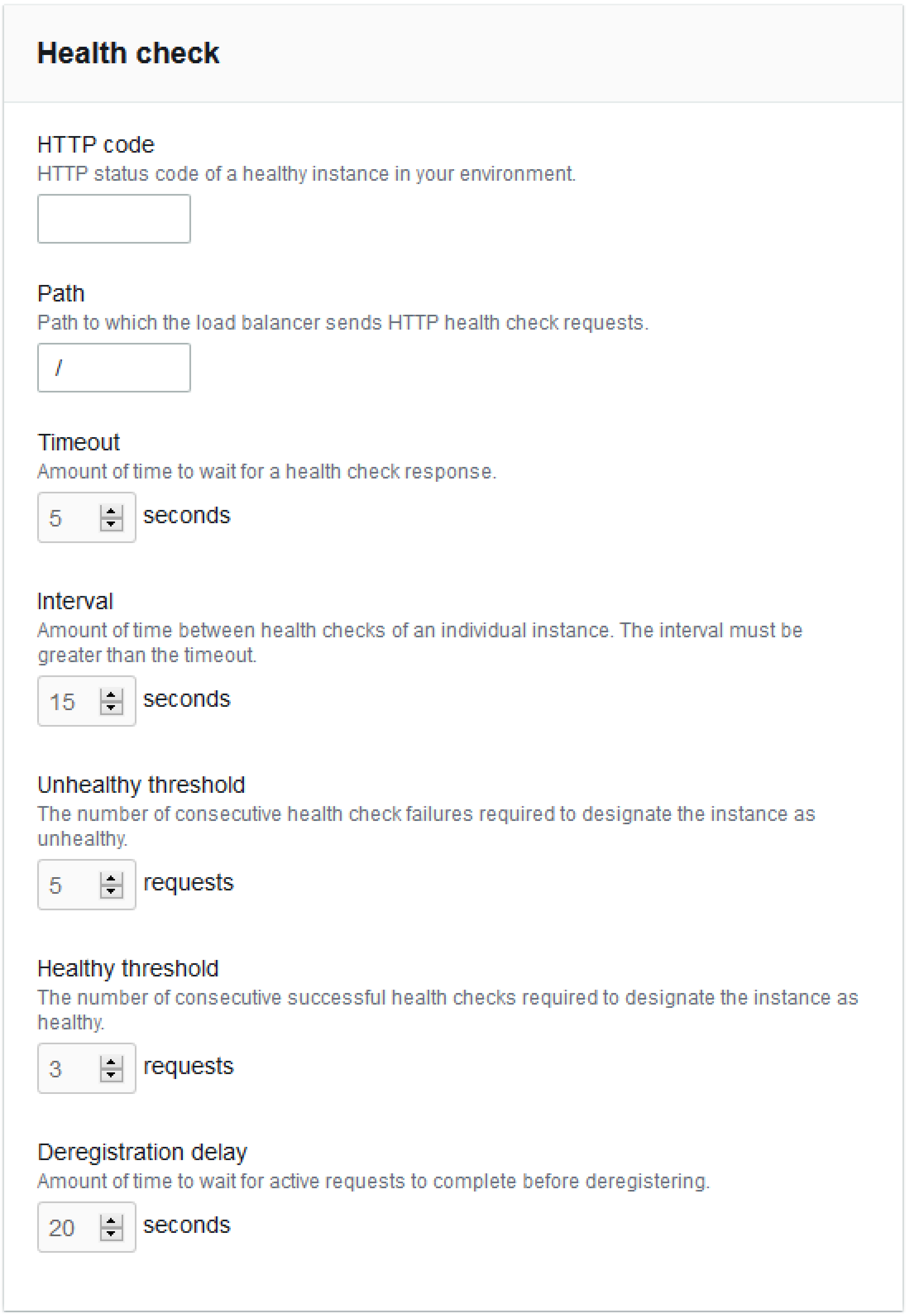
Protocol

HTTP

Health check

Use the following settings to configure process health checks:

- **HTTP code** – The HTTP status code designating a healthy process.
- **Path** – The health check request path for the process.
- **Timeout** – The amount of time, in seconds, to wait for a health check response.
- **Interval** – The amount of time, in seconds, between health checks of an individual instance. The interval must be greater than the timeout.
- **Unhealthy threshold**, **Healthy threshold** – The number of health checks that must fail or pass, respectively, before Elastic Load Balancing changes an instance's health state.
- **Deregistration delay** – The amount of time, in seconds, to wait for active requests to complete before deregistering an instance.



Health check

HTTP code

HTTP status code of a healthy instance in your environment.

Path

Path to which the load balancer sends HTTP health check requests.

/

Timeout

Amount of time to wait for a health check response.

5 seconds

Interval

Amount of time between health checks of an individual instance. The interval must be greater than the timeout.

15 seconds

Unhealthy threshold

The number of consecutive health check failures required to designate the instance as unhealthy.

5 requests

Healthy threshold

The number of consecutive successful health checks required to designate the instance as healthy.

3 requests

Deregistration delay

Amount of time to wait for active requests to complete before deregistering.

20 seconds

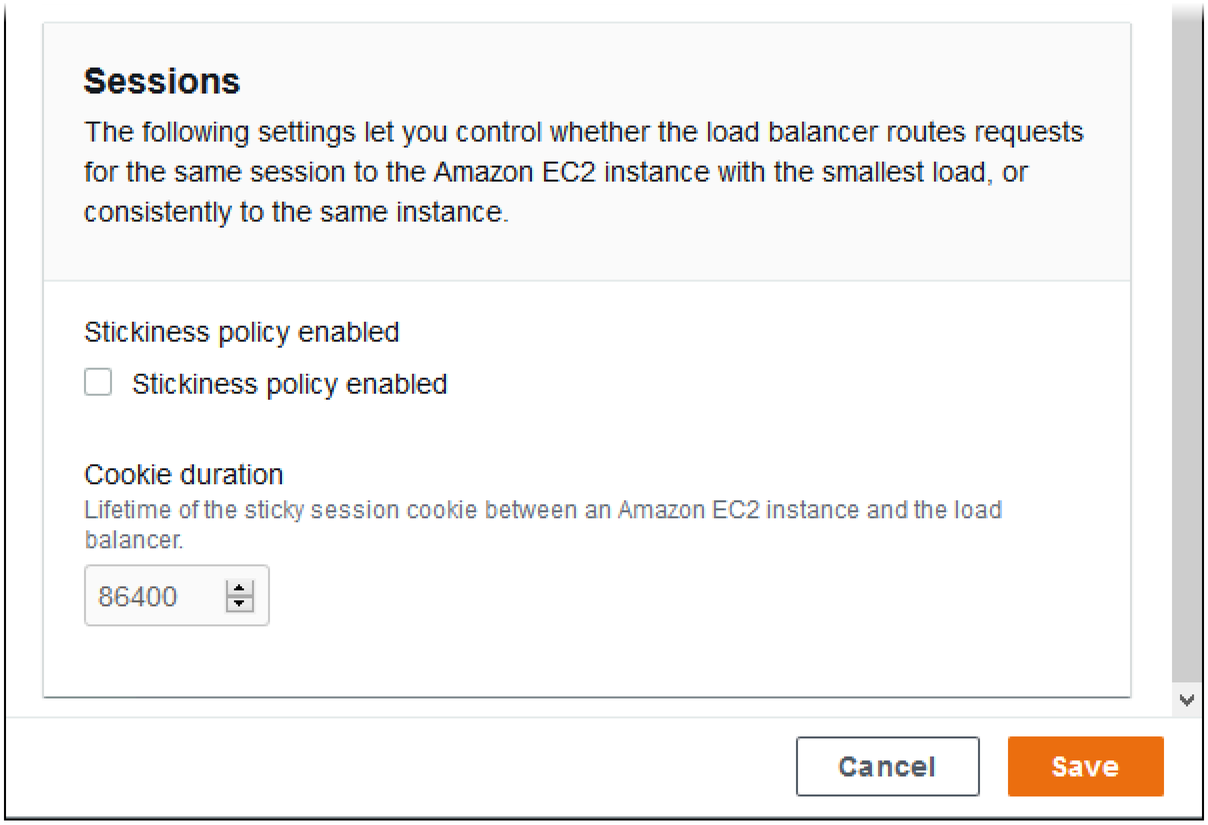
Note

The Elastic Load Balancing health check doesn't affect the health check behavior of an environment's Auto Scaling group. Instances that fail an Elastic Load Balancing health check are not automatically replaced by Amazon EC2 Auto Scaling unless you manually configure Amazon EC2 Auto Scaling to do so. See Auto Scaling health check setting for your Elastic Beanstalk environment for details.

For more information about health checks and how they influence your environment's overall health, see Basic health reporting.

Sessions

Select or clear the **Stickiness policy enabled** box to enable or disable sticky sessions. Use **Cookie duration** to configure a sticky session's duration, up to **`604800`** seconds.



Sessions

The following settings let you control whether the load balancer routes requests for the same session to the Amazon EC2 instance with the smallest load, or consistently to the same instance.

Stickiness policy enabled

☐ Stickiness policy enabled

Cookie duration

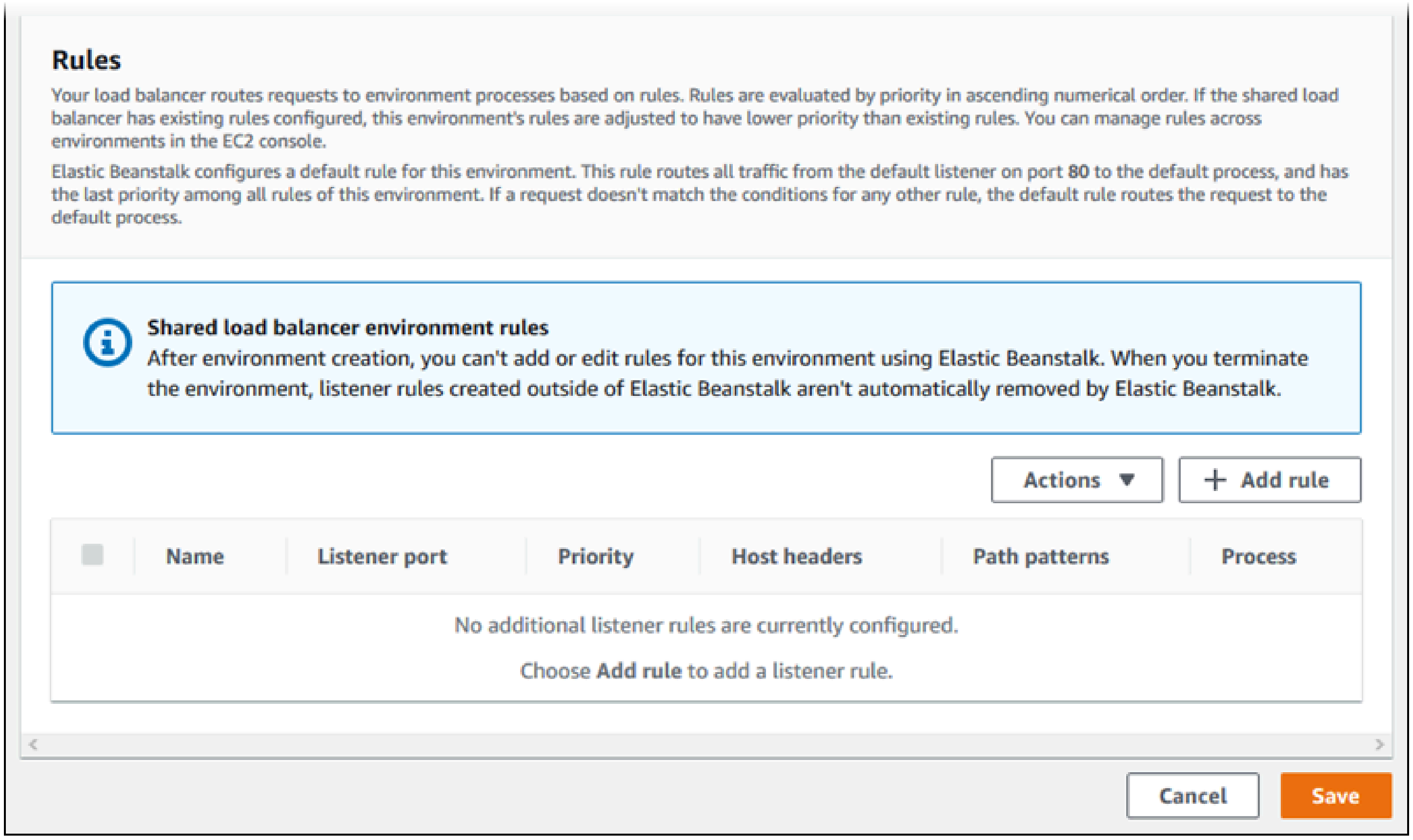
Lifetime of the sticky session cookie between an Amazon EC2 instance and the load balancer.

86400

Rules

Use this list to specify custom listener rules for your shared load balancer. A rule maps requests that the listener receives on a specific path pattern to a target process. Each listener can have multiple rules, routing requests on different paths to different processes on instances of the different environments sharing the listener.

Rules have numeric priorities that determine the precedence in which they are applied to incoming requests. Elastic Beanstalk adds a default rule that routes all the default listener's traffic to the default process of your new environment. The default rule's precedence is the lowest; it's applied if no other rule for the same listener matches the incoming request. Initially, if you haven't added custom rules, the list is empty. The default rule isn't displayed.



Rules

Your load balancer routes requests to environment processes based on rules. Rules are evaluated by priority in ascending numerical order. If the shared load balancer has existing rules configured, this environment's rules are adjusted to have lower priority than existing rules. You can manage rules across environments in the EC2 console.

Elastic Beanstalk configures a default rule for this environment. This rule routes all traffic from the default listener on port **80** to the default process, and has the last priority among all rules of this environment. If a request doesn't match the conditions for any other rule, the default rule routes the request to the default process.

Shared load balancer environment rules
After environment creation, you can't add or edit rules for this environment using Elastic Beanstalk. When you terminate the environment, listener rules created outside of Elastic Beanstalk aren't automatically removed by Elastic Beanstalk.

	Name	Listener port	Priority	Host headers	Path patterns	Process

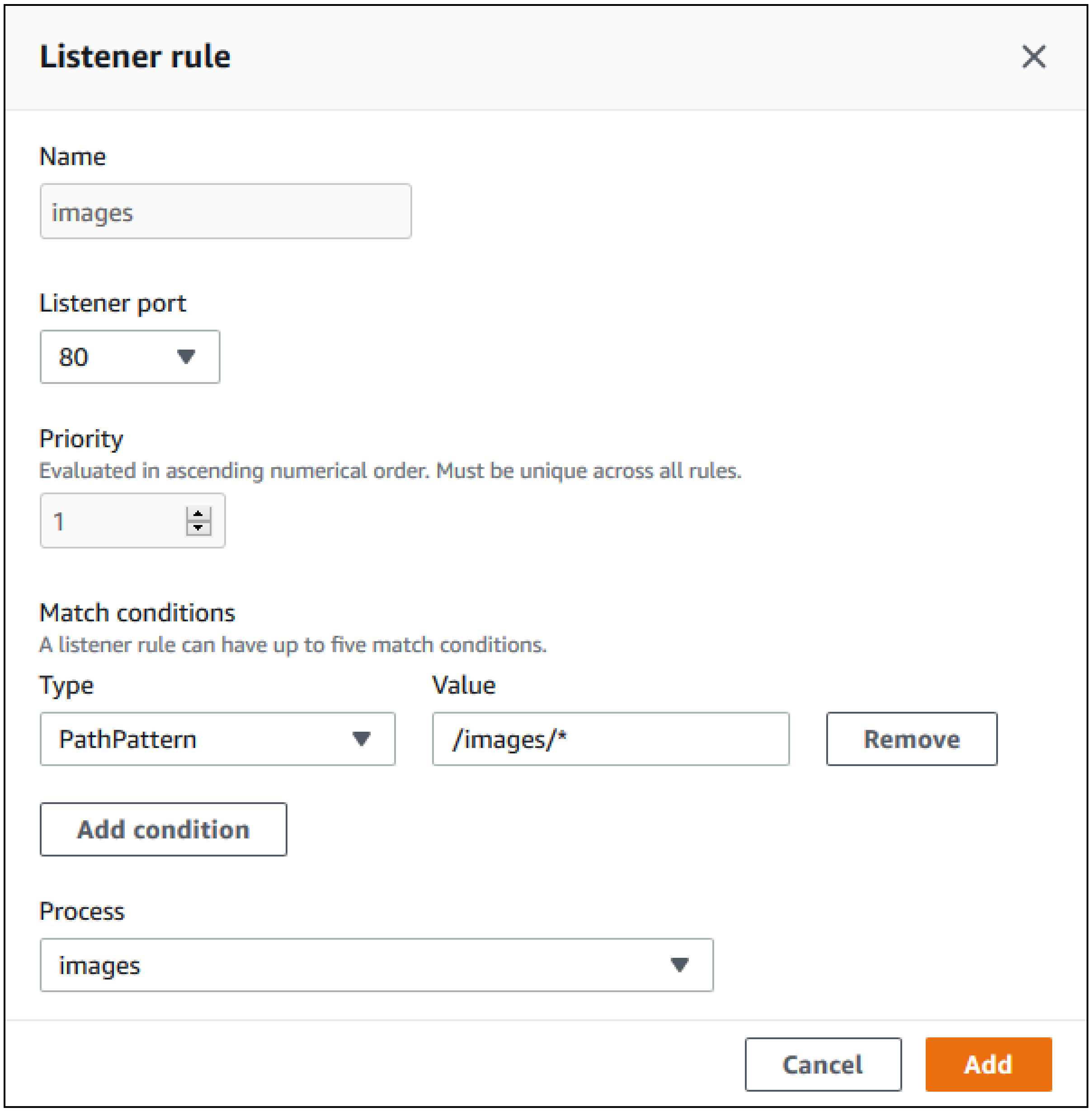
No additional listener rules are currently configured.

Choose **Add rule** to add a listener rule.

You can edit the settings of an existing rule, or add a new rule. To start editing a rule on the list or adding a rule to it, use the same steps listed for the process list. The **Listener rule** dialog box opens, with the following settings:

- **Name** – The rule's name.
- **Listener port** – The port of the listener that the rule applies to.
- **Priority** – The rule's priority. A lower priority number has higher precedence. Priorities of a listener's rules must be unique. Elastic Beanstalk treats rule priorities as relative across sharing environments, and maps them to absolute priorities during creation.

- **Match conditions** – A list of request URL conditions that the rule applies to. There are two types of conditions: **HostHeader** (the URL's domain part), and **PathPattern** (the URL's path part). One condition is reserved for the environment subdomain, and you can add up to four conditions. Each condition value is up to 128 characters in length, and can include wildcard characters.
- **Process** – The process to which the load balancer routes requests that match the rule.



Listener rule

Name: images

Listener port: 80

Priority
Evaluated in ascending numerical order. Must be unique across all rules.
1

Match conditions
A listener rule can have up to five match conditions.

Type	Value
PathPattern	/images/*

Process: images

Example: use a shared Application Load Balancer for a secure micro-service-based application

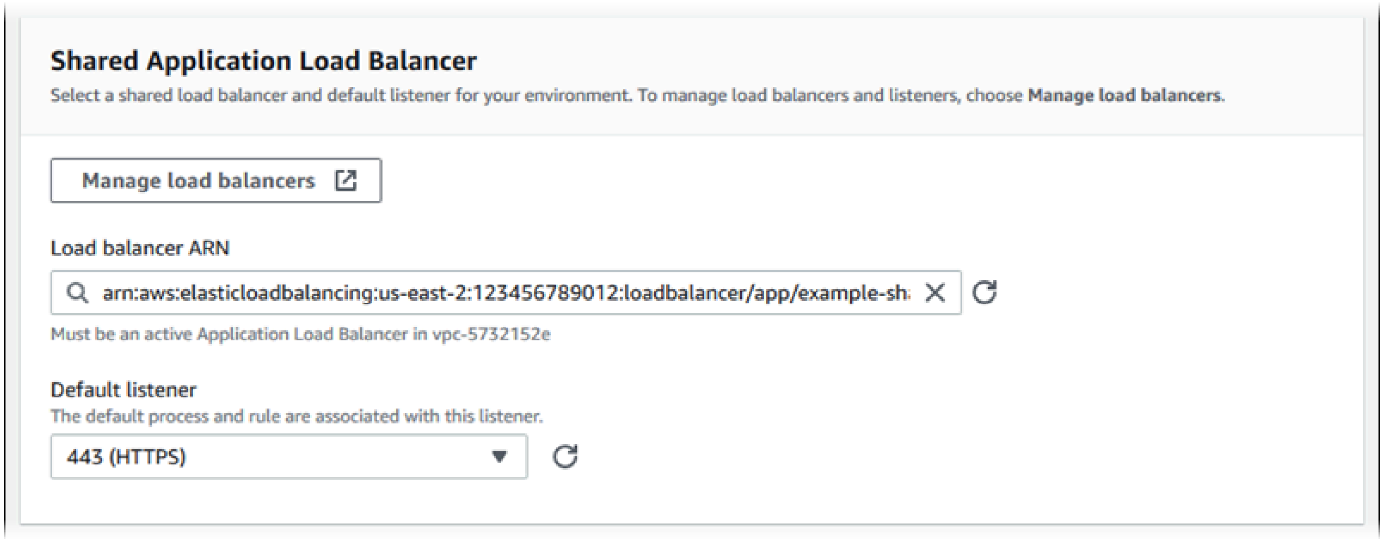
In this example, your application consists of several micro services, each implemented as an Elastic Beanstalk environment. In addition, you require end-to-end traffic encryption. We'll demonstrate one of the micro-service environments, which has a main process for user requests and a separate process for handling administrative requests.

To meet these requirements, use Amazon EC2 to create an Application Load Balancer that you'll share among your micro services. Add a secure listener on port 443 and the HTTPS protocol. Then add multiple SSL certificates to the listener—one per micro-service domain. For details about creating the Application Load Balancer and secure listener, see Create an Application Load Balancer and Create an HTTPS listener for your Application Load Balancer in the *User Guide for Application Load Balancers*.

In Elastic Beanstalk, configure each micro-service environment to use the shared Application Load Balancer and set the default listener port to 443. In the case of the particular environment that we're demonstrating here, indicate that the default process listens to port 443 on HTTPS, and add a process and a listener rule for admin traffic on a different path.

To configure the shared load balancer for this example

1. In the **Shared Application Load Balancer** section, select your load balancer, and then, for **Default listener port**, select **443**. The listener port should already be selected if it's the only listener that the load balancer has.



Shared Application Load Balancer

Select a shared load balancer and default listener for your environment. To manage load balancers and listeners, choose **Manage load balancers**.

Load balancer ARN

arn:aws:elasticloadbalancing:us-east-2:123456789012:loadbalancer/app/example-sh

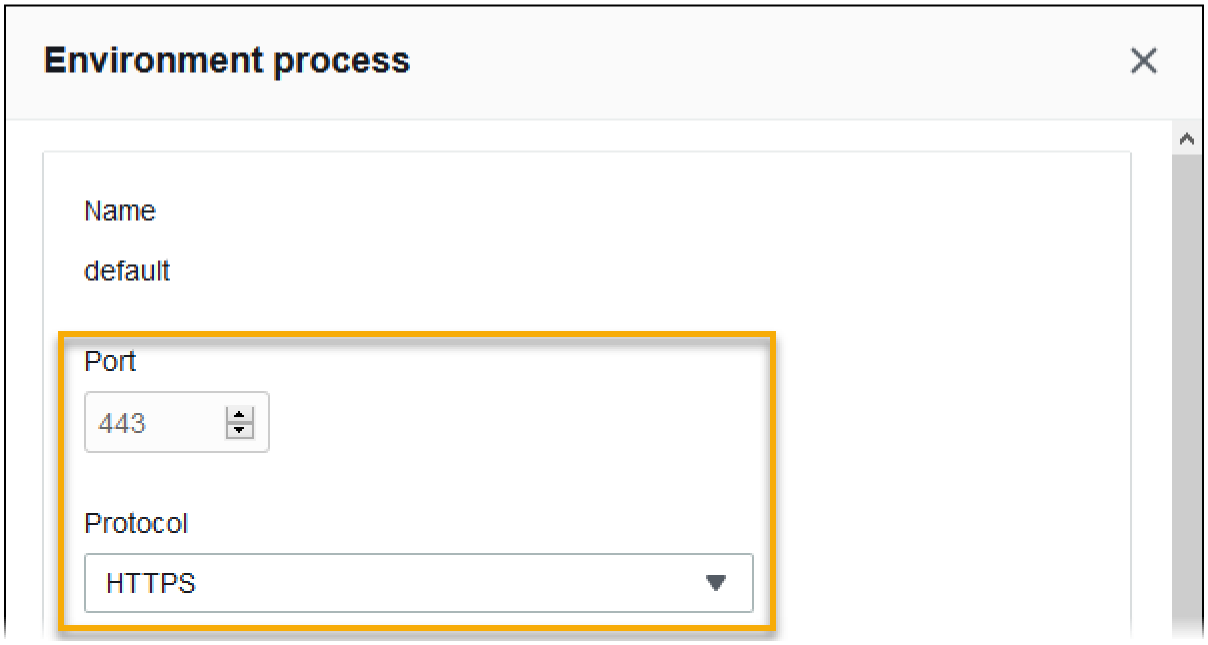
Must be an active Application Load Balancer in vpc-5732152e

Default listener

The default process and rule are associated with this listener.

443 (HTTPS)

2. *Configure the default process to HTTPS.* Select the default process, and then for **Actions**, choose **Edit**. For **Port**, enter **443**. For **Protocol**, select **HTTPS**.



Environment process

Name

default

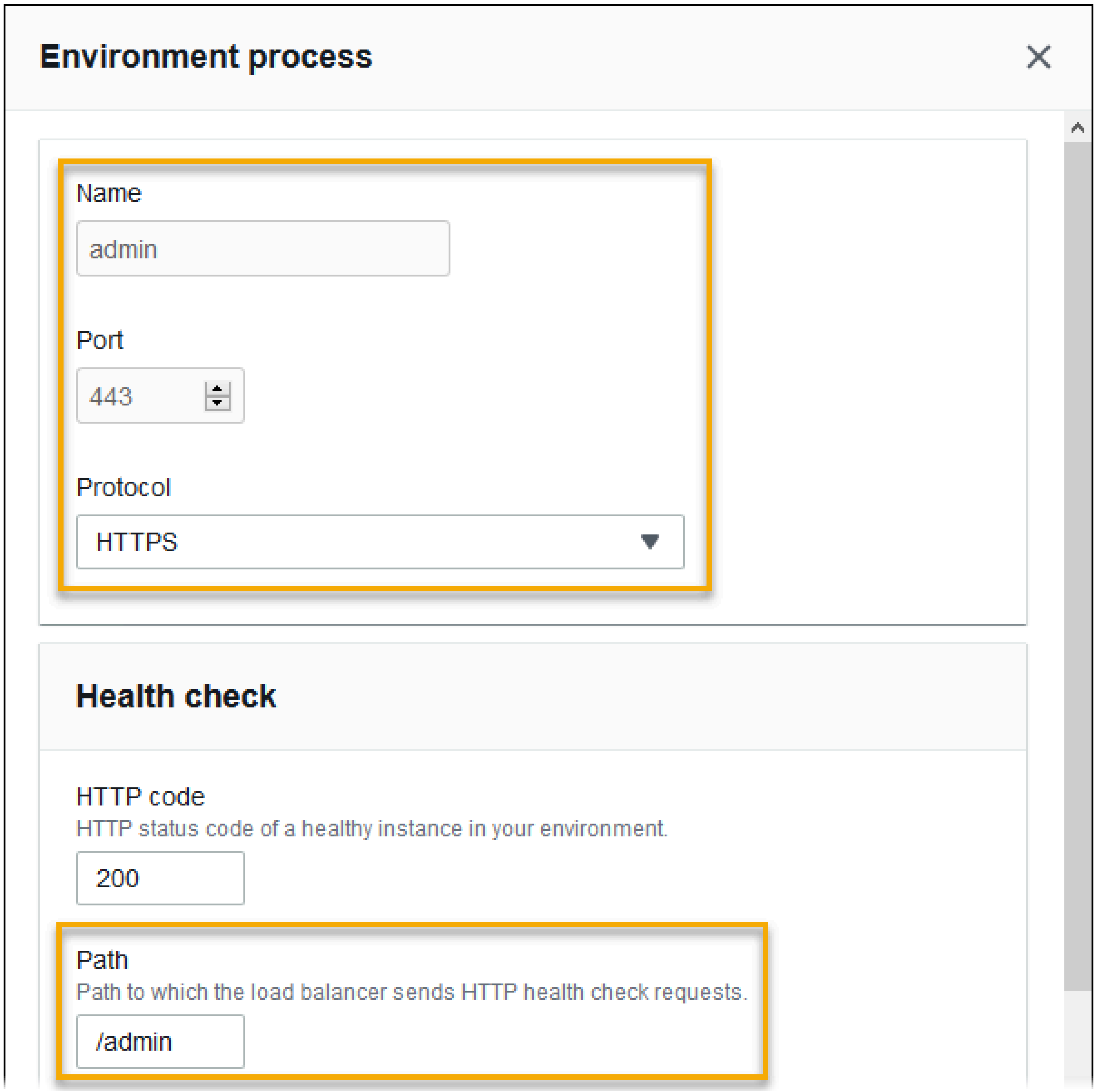
Port

443

Protocol

HTTPS

3. *Add an admin process.* For **Name**, enter **`admin`**. For **Port**, enter **`443`**. For **Protocol**, select **HTTPS**. Under **Health check**, for **Path** enter **`/admin`**.


Environment process

Name
admin

Port
443

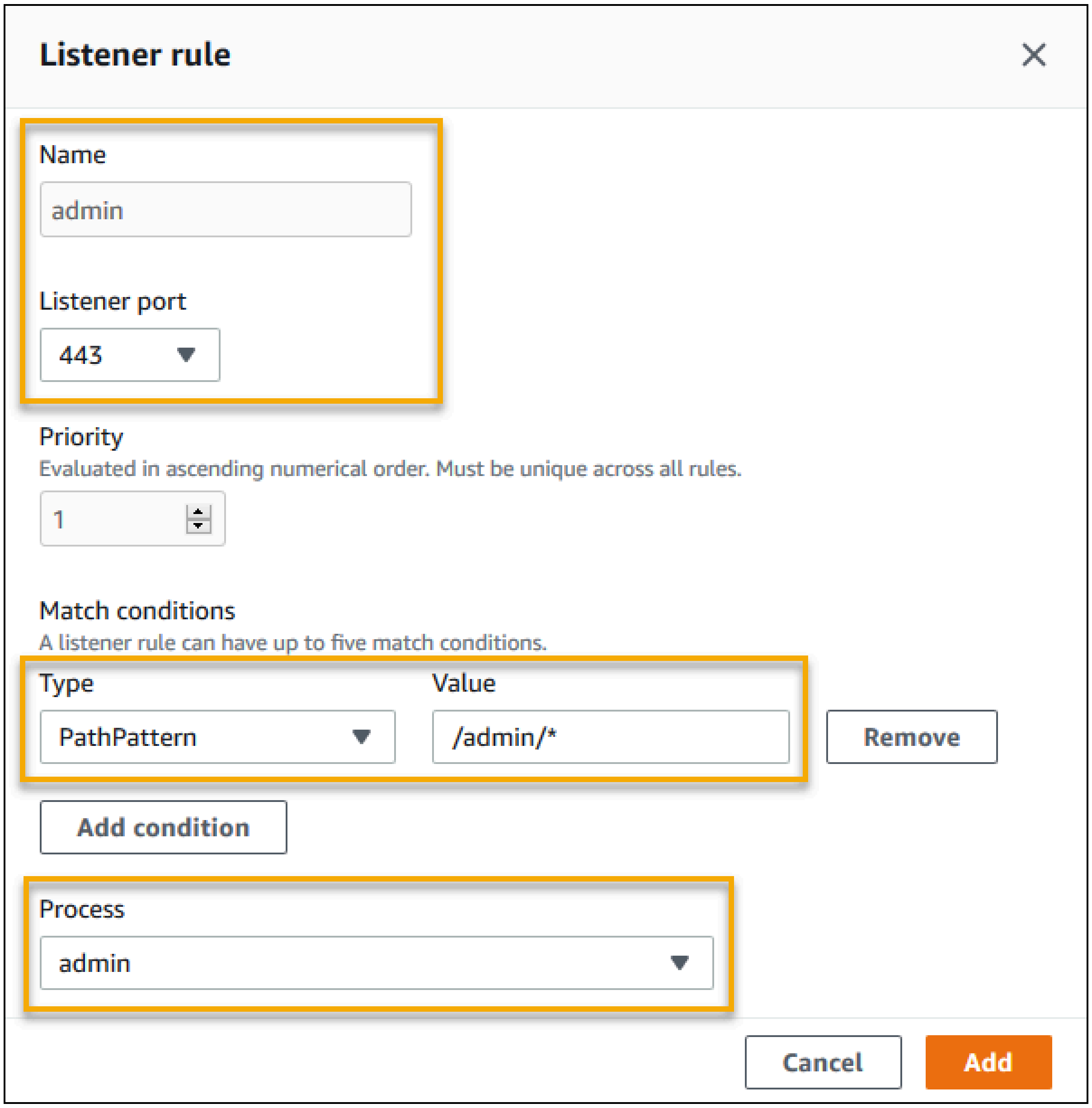
Protocol
HTTPS

Health check

HTTP code
HTTP status code of a healthy instance in your environment.
200

Path
Path to which the load balancer sends HTTP health check requests.
/admin

4. *Add a rule for admin traffic.* For **Name**, enter **`admin`**. For **Listener port**, enter **`443`**. For **Match conditions**, add a **PathPattern** with the value **`/admin/*`**. For **Process**, select **`admin`**.


Listener rule

Name: admin

Listener port: 443

Priority
Evaluated in ascending numerical order. Must be unique across all rules.
1

Match conditions
A listener rule can have up to five match conditions.

Type	Value
PathPattern	/admin/*

Process: admin

Configuring a shared Application Load Balancer using the EB CLI

The EB CLI prompts you to choose a load balancer type when you run eb create. If you choose `application` (the default), and if your account has at least one sharable Application Load Balancer, the EB CLI also asks you if you want to use a shared Application Load Balancer. If you answer **y**, you are also prompted to select the load balancer and default port.

```
$ eb create
Enter Environment Name
(default is my-app): test-env
```

```
Enter DNS CNAME prefix
(default is my-app): test-env-DLW24ED23SF

Select a load balancer type
1) classic
2) application
3) network
(default is 2):

Your account has one or more sharable load balancers. Would you like your new
 environment to use a shared load balancer?(y/N) y

Select a shared load balancer
1)MySharedALB1 - arn:aws:elasticloadbalancing:us-east-2:123456789012:loadbalancer/app/
MySharedALB1/6d69caa75b15d46e
2)MySharedALB2 - arn:aws:elasticloadbalancing:us-east-2:123456789012:loadbalancer/app/
MySharedALB2/e574ea4c37ad2ec8
(default is 1): 2

Select a listener port for your shared load balancer
1) 80
2) 100
3) 443
(default is 1): 3
```

You can also specify a shared load balancer using command options.

```
$ eb create test-env --elb-type application --shared-lb MySharedALB2 --shared-lb-
port 443
```

Shared Application Load Balancer namespaces

You can find settings related to shared Application Load Balancers in the following namespaces:

- `aws:elasticbeanstalk:environment` – Choose the load balancer type for the environment, and tell Elastic Beanstalk that you'll use a shared load balancer.

  You can't set these two options in configuration files (.Ebextensions).
- `aws:elbv2:loadbalancer` – Configure the shared Application Load Balancer ARN and security groups.

- aws:elbv2:listener – Associate listeners of the shared Application Load Balancer with environment processes by listing listener rules.
- aws:elbv2:listenerrule – Configure listener rules that route traffic to different processes, depending on the request path. Rules are unique to Application Load Balancers—both dedicated and shared.
- aws:elasticbeanstalk:environment:process – Configure health checks and specify the port and protocol for the processes that run on your environment's instances.

Example .ebextensions/application-load-balancer-shared.config

To get started with a shared Application Load Balancer, use the Elastic Beanstalk console, EB CLI, or API to set the load balancer type to `application` and choose to use a shared load balancer. Use a configuration file to configure the shared load balancer.

```
option_settings:
  aws:elbv2:loadbalancer:
    SharedLoadBalancer: arn:aws:elasticloadbalancing:us-
east-2:123456789012:loadbalancer/app/MySharedALB2/e574ea4c37ad2ec8
```

Note

You can configure this option only during environment creation.

Example .ebextensions/alb-shared-secure-listener.config

The following configuration file selects a default secure listener on port 443 for the shared load balancer, and sets the default process to listen to port 443.

```
option_settings:
  aws:elbv2:loadbalancer:
    SharedLoadBalancer: arn:aws:elasticloadbalancing:us-
east-2:123456789012:loadbalancer/app/MySharedALB2/e574ea4c37ad2ec8
  aws:elbv2:listener:443:
    rules: default
  aws:elasticbeanstalk:environment:process:default:
    Port: '443'
    Protocol: HTTPS
```

Example .ebextensions/alb-shared-admin-rule.config

The following configuration file builds on the previous example and adds a rule that routes traffic with a request path of `/admin` to a process named `admin` that listens on port 4443.

```
option_settings:
  aws:elbv2:loadbalancer:
    SharedLoadBalancer: arn:aws:elasticloadbalancing:us-
east-2:123456789012:loadbalancer/app/MySharedALB2/e574ea4c37ad2ec8
  aws:elbv2:listener:443:
    rules: default,admin
  aws:elasticbeanstalk:environment:process:default:
    Port: '443'
    Protocol: HTTPS
  aws:elasticbeanstalk:environment:process:admin:
    HealthCheckPath: /admin
    Port: '4443'
    Protocol: HTTPS
  aws:elbv2:listenerrule:admin:
    PathPatterns: /admin/*
    Priority: 1
    Process: admin
```

Configuring a Network Load Balancer

When you enable load balancing, your AWS Elastic Beanstalk environment is equipped with an Elastic Load Balancing load balancer to distribute traffic among the instances in your environment. Elastic Load Balancing supports several load balancer types. To learn about them, see the Elastic Load Balancing User Guide. Elastic Beanstalk can create a load balancer for you, or let you specify a shared load balancer that you've created.

This topic describes the configuration of a Network Load Balancer that Elastic Beanstalk creates and dedicates to your environment. For information about configuring all the load balancer types that Elastic Beanstalk supports, see Load balancer for your Elastic Beanstalk environment.

Note

You can choose the type of load balancer that your environment uses only during environment creation. You can change settings to manage the behavior of your running environment's load balancer, but you can't change its type.

Introduction

With a Network Load Balancer, the default listener accepts TCP requests on port 80 and distributes them to the instances in your environment. You can configure health check behavior, configure the listener port, or add a listener on another port.

Note

Unlike a Classic Load Balancer or an Application Load Balancer, a Network Load Balancer can't have application layer (layer 7) HTTP or HTTPS listeners. It only supports transport layer (layer 4) TCP listeners. HTTP and HTTPS traffic can be routed to your environment over TCP. To establish secure HTTPS connections between web clients and your environment, install a self-signed certificate on the environment's instances, and configure the instances to listen on the appropriate port (typically 443) and terminate HTTPS connections. The configuration varies per platform. See Configuring HTTPS Termination at the instance for instructions. Then configure your Network Load Balancer to add a listener that maps to a process listening on the appropriate port.

A Network Load Balancer supports active health checks. These checks are based on messages to the root (/) path. In addition, a Network Load Balancer supports passive health checks. It automatically detects faulty backend instances and routes traffic only to healthy instances.

Configuring a Network Load Balancer using the Elastic Beanstalk console

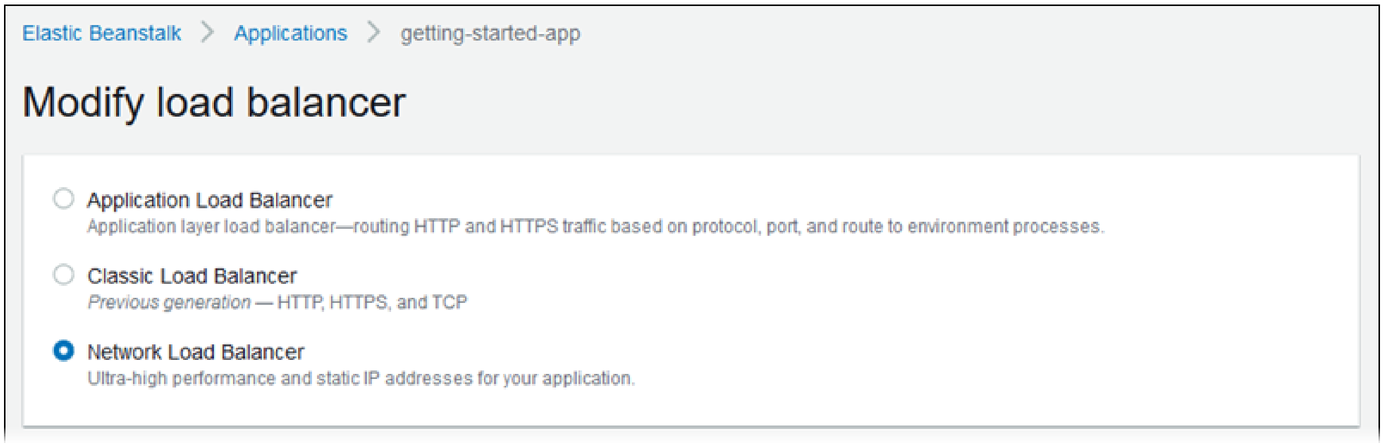
You can use the Elastic Beanstalk console to configure a Network Load Balancer's listeners and processes during environment creation, or later when your environment is running.

To configure a Network Load Balancer in the Elastic Beanstalk console during environment creation

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**.
3. Choose Create a new environment to start creating your environment.
4. On the wizard's main page, before choosing **Create environment**, choose **Configure more options**.
5. Choose the **High availability** configuration preset.

Alternatively, in the **Capacity** configuration category, configure a **Load balanced** environment type. For details, see [Capacity](#).

6. In the **Load balancer** configuration category, choose **Edit**.
7. Select the **Network Load Balancer** option, if it isn't already selected.



Modify load balancer

( ) Application Load Balancer
Application layer load balancer—routing HTTP and HTTPS traffic based on protocol, port, and route to environment processes.

( ) Classic Load Balancer
Previous generation — HTTP, HTTPS, and TCP

(✓) Network Load Balancer
Ultra-high performance and static IP addresses for your application.

8. Make any Network Load Balancer configuration changes that your environment requires.
9. Choose **Save**, and then make any other configuration changes that your environment requires.
10. Choose **Create environment**.

To configure a running environment's Network Load Balancer in the Elastic Beanstalk console

1. Open the [Elastic Beanstalk console](#), and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Load balancer** configuration category, choose **Edit**.

> **Note**
>
> If the **Load balancer** configuration category doesn't have an **Edit** button, your environment doesn't have a load balancer. To learn how to set one up, see [Changing environment type](#).

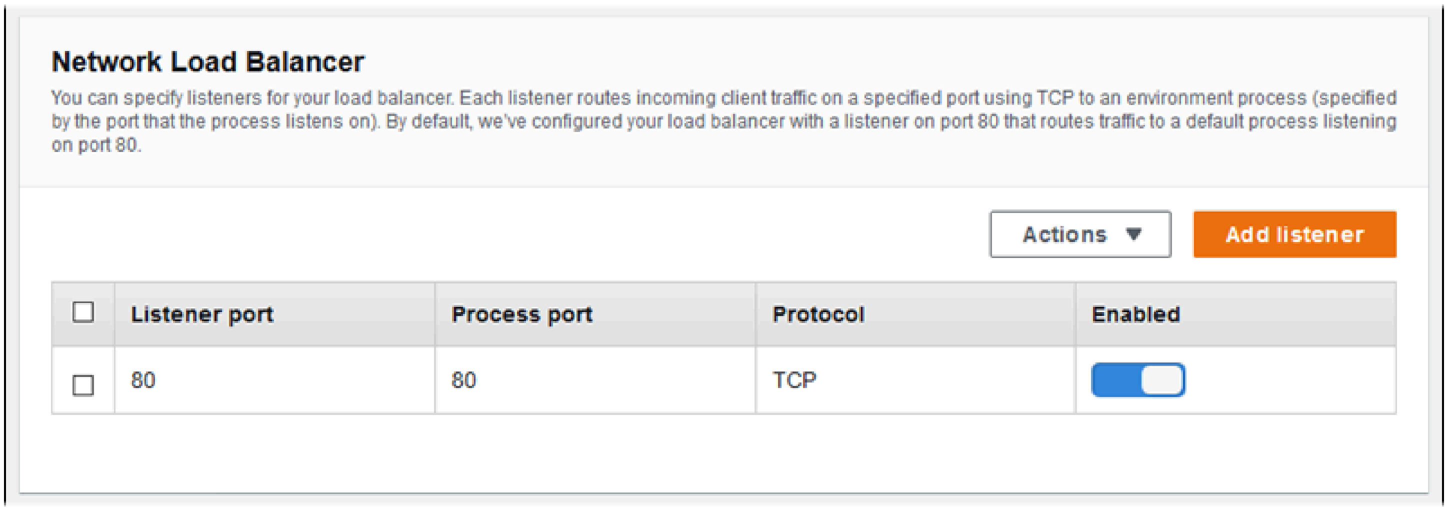
5. Make the Network Load Balancer configuration changes that your environment requires.
6. To save the changes choose **Apply** at the bottom of the page.

Network Load Balancer settings

- Listeners
- Processes

Listeners

Use this list to specify listeners for your load balancer. Each listener routes incoming client traffic on a specified port to a process on your instances. Initially, the list shows the default listener, which routes incoming traffic on port 80 to a process named **default**, which listens to port 80.



Network Load Balancer

You can specify listeners for your load balancer. Each listener routes incoming client traffic on a specified port using TCP to an environment process (specified by the port that the process listens on). By default, we've configured your load balancer with a listener on port 80 that routes traffic to a default process listening on port 80.

☐	Listener port	Process port	Protocol	Enabled
☐	80	80	TCP	●

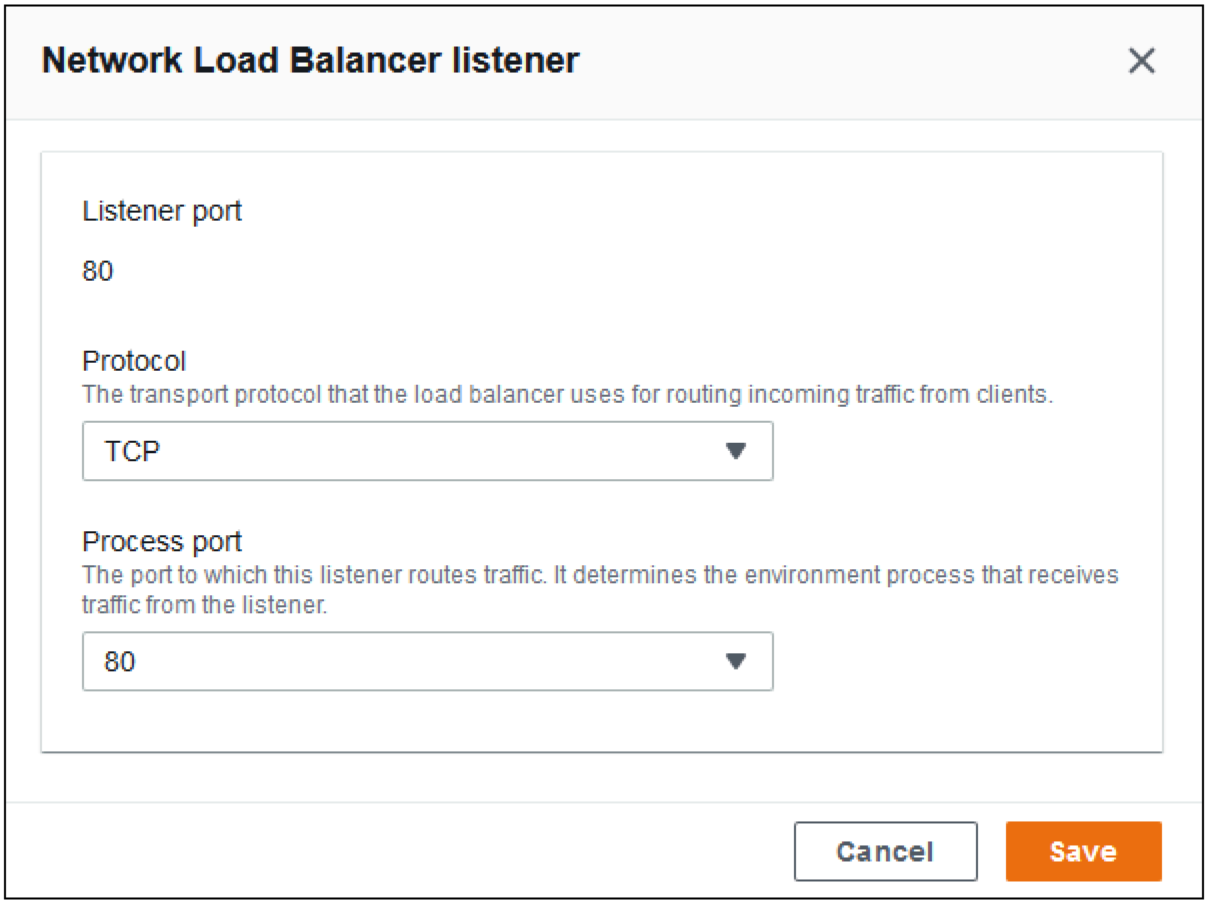
To configure an existing listener

1. Select the check box next to its table entry, and then choose **Actions**, **Edit**.
2. Use the **Network Load Balancer listener** dialog box to edit settings, and then choose **Save**.

To add a listener

1. Choose **Add listener**.
2. In the **Network Load Balancer listener** dialog box, configure the required settings, and then choose **Add**.

Use the **Network Load Balancer listener** dialog box to configure the port on which the listener listens to traffic, and to choose the process to which you want to route traffic (specified by the port that the process listens to).



Network Load Balancer listener

Listener port

80

Protocol

The transport protocol that the load balancer uses for routing incoming traffic from clients.

TCP

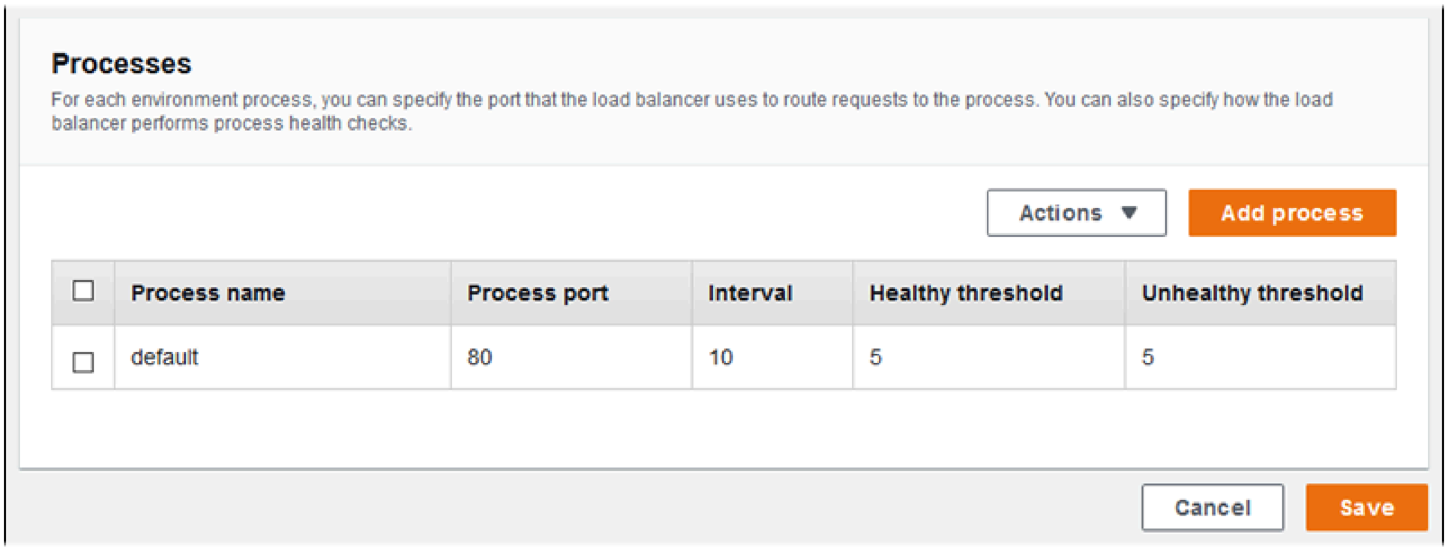
Process port

The port to which this listener routes traffic. It determines the environment process that receives traffic from the listener.

80

Processes

Use this list to specify processes for your load balancer. A process is a target for listeners to route traffic to. Each listener routes incoming client traffic on a specified port to a process on your instances. Initially, the list shows the default process, which listens to incoming traffic on port 80.



Processes

For each environment process, you can specify the port that the load balancer uses to route requests to the process. You can also specify how the load balancer performs process health checks.

☐	Process name	Process port	Interval	Healthy threshold	Unhealthy threshold
☐	default	80	10	5	5

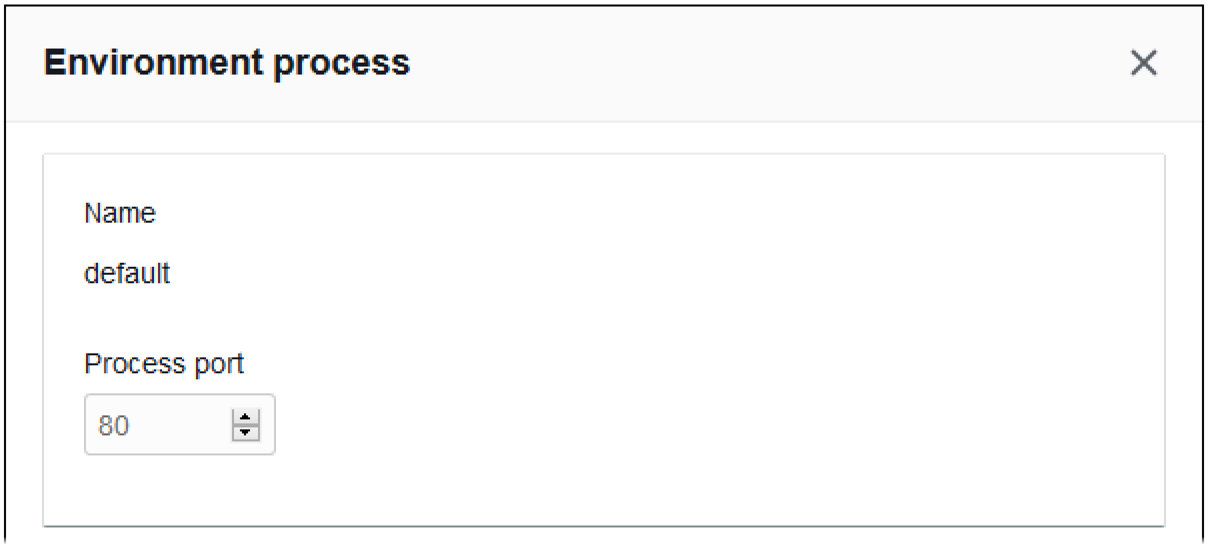
You can edit the settings of an existing process, or add a new process. To start editing a process on the list or adding a process to it, use the same steps listed for the listener list. The **Environment process** dialog box opens.

Network Load Balancer's environment process dialog box settings

- Definition
- Health check

Definition

Use these settings to define the process: its **Name** and the **Process port** on which it listens to requests.


Environment process

Name

default

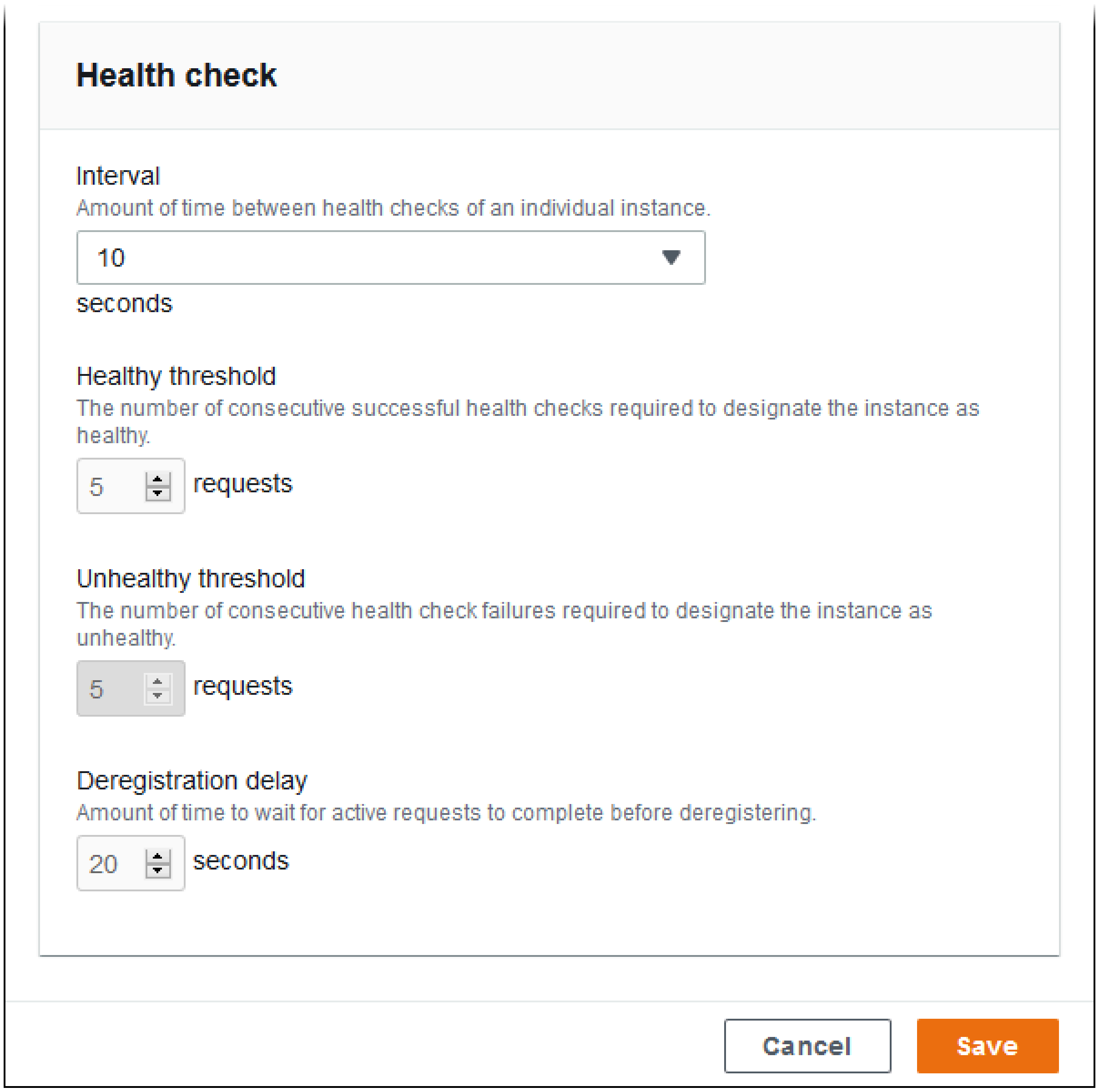
Process port

80

Health check

Use the following settings to configure process health checks:

- **Interval** – The amount of time, in seconds, between health checks of an individual instance.
- **Healthy threshold** – The number of health checks that must pass before Elastic Load Balancing changes an instance's health state. (For Network Load Balancer, **Unhealthy threshold** is a read-only setting that is always equal to the healthy threshold value.)
- **Deregistration delay** – The amount of time, in seconds, to wait for active requests to complete before deregistering an instance.



Health check

Interval
Amount of time between health checks of an individual instance.
10
seconds

Healthy threshold
The number of consecutive successful health checks required to designate the instance as healthy.
5 requests

Unhealthy threshold
The number of consecutive health check failures required to designate the instance as unhealthy.
5 requests

Deregistration delay
Amount of time to wait for active requests to complete before deregistering.
20 seconds

Note

The Elastic Load Balancing health check doesn't affect the health check behavior of an environment's Auto Scaling group. Instances that fail an Elastic Load Balancing health check will not automatically be replaced by Amazon EC2 Auto Scaling unless you manually configure Amazon EC2 Auto Scaling to do so. See Auto Scaling health check setting for your Elastic Beanstalk environment for details.

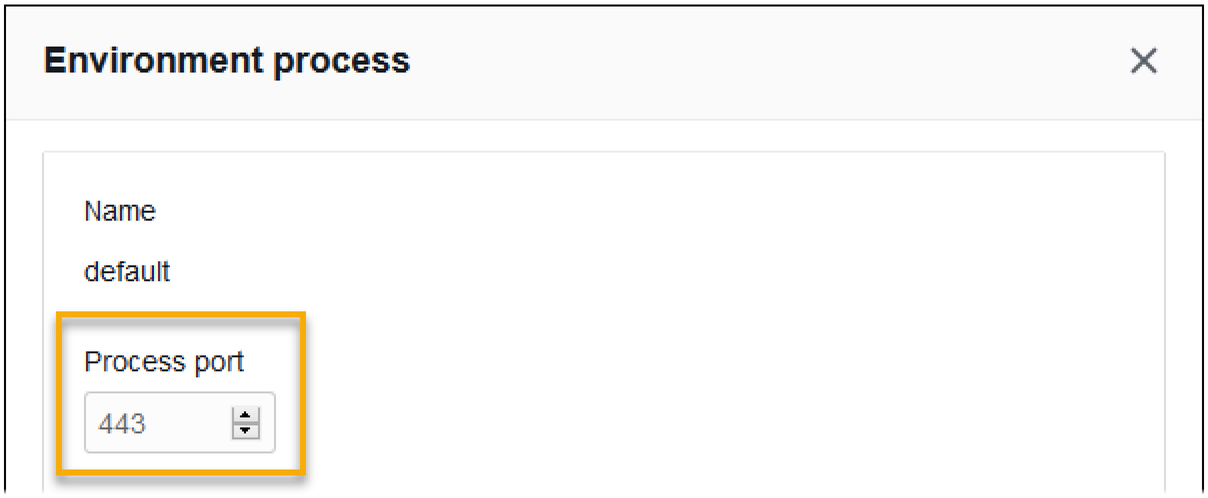
For more information about health checks and how they influence your environment's overall health, see Basic health reporting.

Example: Network Load Balancer for an environment with end-to-end encryption

In this example, your application requires end-to-end traffic encryption. To configure your environment's Network Load Balancer to meet these requirements, you configure the default process to listen to port 443, add a listener to port 443 that routes traffic to the default process, and disable the default listener.

To configure the load balancer for this example

1. *Configure the default process.* Select the default process, and then, for **Actions**, choose **Edit**. For **Process port**, type 443.

   Environment process

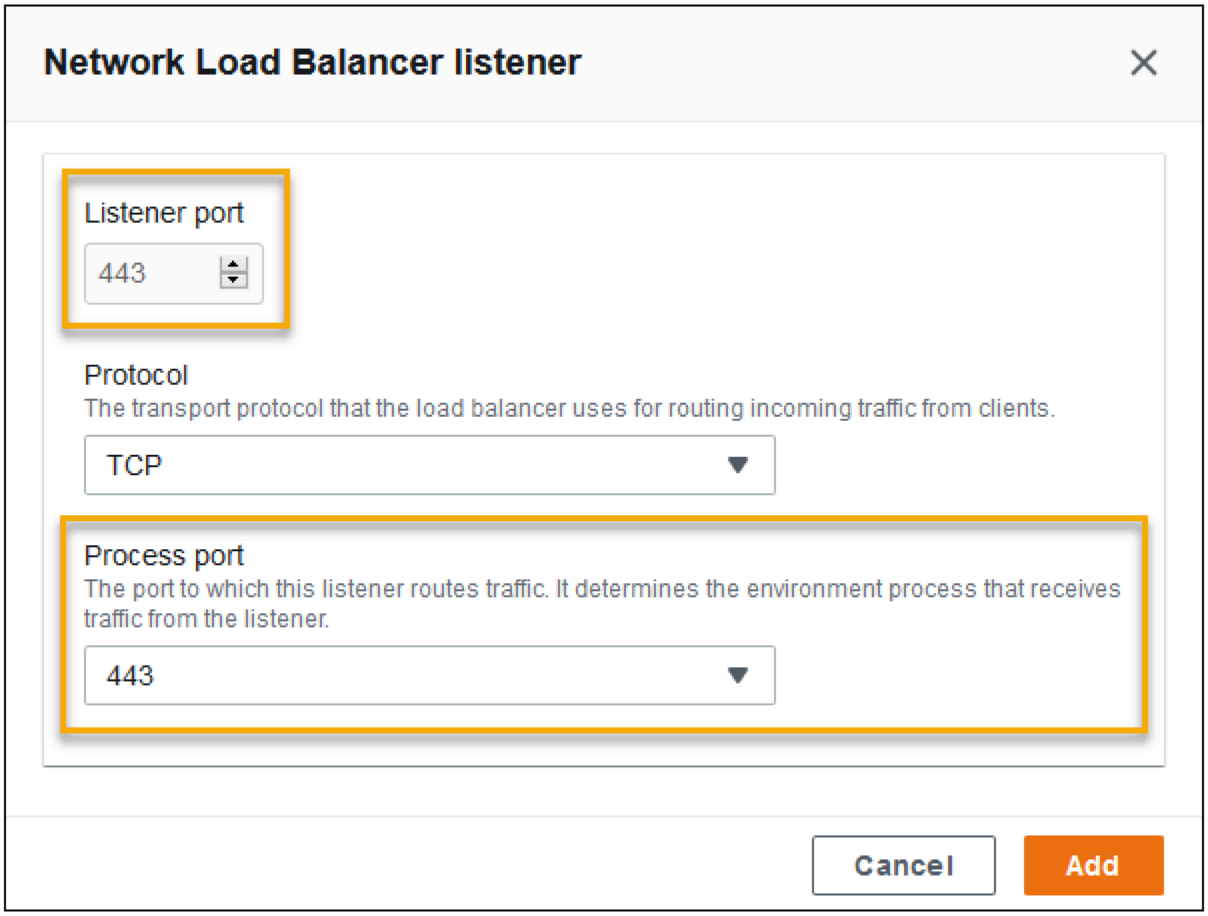
   Name

   default

   Process port

   443

2. *Add a port 443 listener.* Add a new listener. For **Listener port**, type 443. For **Process port**, make sure that 443 is selected.

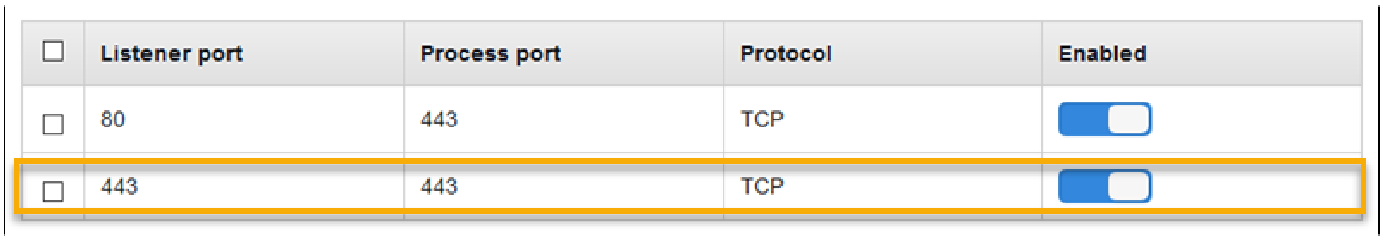


Listener port: 443

Protocol
The transport protocol that the load balancer uses for routing incoming traffic from clients.
TCP

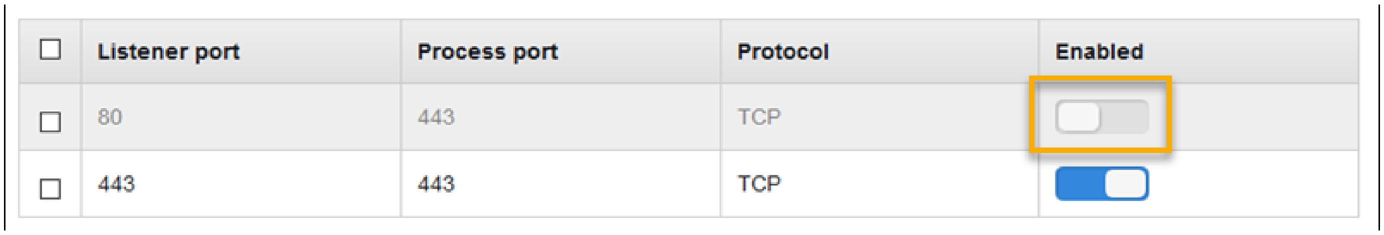
Process port
The port to which this listener routes traffic. It determines the environment process that receives traffic from the listener.
443

You can now see your additional listener on the list.



☐	Listener port	Process port	Protocol	Enabled
☐	80	443	TCP	●
☐	443	443	TCP	●

3. *Disable the default port 80 listener.* For the default listener, turn off the **Enabled** option.



☐	Listener port	Process port	Protocol	Enabled
☐	80	443	TCP	○
☐	443	443	TCP	●

Configuring a Network Load Balancer using the EB CLI

The EB CLI prompts you to choose a load balancer type when you run **eb create**.

```
$ eb create
Enter Environment Name
(default is my-app): test-env
Enter DNS CNAME prefix
(default is my-app): test-env-DLW24ED23SF

Select a load balancer type
1) classic
2) application
3) network
(default is 1): 3
```

You can also specify a load balancer type with the `--elb-type` option.

```
$ eb create test-env --elb-type network
```

Network Load Balancer namespaces

You can find settings related to Network Load Balancers in the following namespaces:

- `aws:elasticbeanstalk:environment` – Choose the load balancer type for the environment. The value for a Network Load Balancer is `network`.
- `aws:elbv2:listener` – Configure listeners on the Network Load Balancer. These settings map to the settings in `aws:elb:listener` for Classic Load Balancers.
- `aws:elasticbeanstalk:environment:process` – Configure health checks and specify the port and protocol for the processes that run on your environment's instances. The port and protocol settings map to the instance port and instance protocol settings in `aws:elb:listener` for a listener on a Classic Load Balancer. Health check settings map to the settings in the `aws:elb:healthcheck` and `aws:elasticbeanstalk:application` namespaces.

Example .ebextensions/network-load-balancer.config

To get started with a Network Load Balancer, use a configuration file to set the load balancer type to `network`.

```
option_settings:
  aws:elasticbeanstalk:environment:
    LoadBalancerType: network
```

Note

You can set the load balancer type only during environment creation.

Example .ebextensions/nlb-default-process.config

The following configuration file modifies health check settings on the default process.

```
option_settings:
  aws:elasticbeanstalk:environment:process:default:
    DeregistrationDelay: '20'
    HealthCheckInterval: '10'
    HealthyThresholdCount: '5'
    UnhealthyThresholdCount: '5'
    Port: '80'
    Protocol: TCP
```

Example .ebextensions/nlb-secure-listener.config

The following configuration file adds a listener for secure traffic on port 443 and a matching target process that listens to port 443.

```
option_settings:
  aws:elbv2:listener:443:
    DefaultProcess: https
    ListenerEnabled: 'true'
  aws:elasticbeanstalk:environment:process:https:
    Port: '443'
```

The `DefaultProcess` option is named this way because of Application Load Balancers, which can have non-default listeners on the same port for traffic to specific paths (see Application Load Balancer for details). For a Network Load Balancer the option specifies the only target process for this listener.

In this example, we named the process `https` because it listens to secure (HTTPS) traffic. The listener sends traffic to the process on the designated port using the TCP protocol, because a

Network Load Balancer works only with TCP. This is okay, because network traffic for HTTP and HTTPS is implemented on top of TCP.

Configuring access logs

You can use configuration files to configure your environment's load balancer to upload access logs to an Amazon S3 bucket. See the following example configuration files on GitHub for instructions:

- `loadbalancer-accesslogs-existingbucket.config` – Configure the load balancer to upload access logs to an existing Amazon S3 bucket.
- `loadbalancer-accesslogs-newbucket.config` – Configure the load balancer to upload access logs to a new bucket.

Adding a database to your Elastic Beanstalk environment

Elastic Beanstalk provides *coupled* database integration with Amazon Relational Database Service (Amazon RDS). You can use Elastic Beanstalk to add a MySQL, PostgreSQL, Oracle, or SQL Server database to an existing environment or a new one when you create it. When you add a database instance that's coupled to an environment, Elastic Beanstalk provides the connection information to your application. It does this by setting the environment properties for the database hostname, port, user name, password, and database name.

Advantages of using a coupled database

If you haven't used a database instance with your application before, we recommend that you first use the process described in this topic to add a database to a test environment using the Elastic Beanstalk service. By doing this, you can verify that your application can read the environment properties, construct a connection string, and connect to a database instance, without the additional configuration work required for a database external to Elastic Beanstalk.

Considerations when going to production

After you verify that your application works correctly with the database, you may consider moving towards a production environment. At this point you have the option to decouple the database from your Elastic Beanstalk environment to move towards a configuration that offers greater flexibility. The decoupled database can remain operational as an external Amazon RDS database instance. The health of the environment isn't affected by decoupling the database. If you need to terminate the environment, you can do so and also choose the option to keep the database available and operational outside of Elastic Beanstalk.

Advantages of moving to a decoupled database

Using an external database has several advantages. You can connect to the external database from multiple environments, use database types that aren't supported with integrated databases, and perform blue/green deployments. As an alternative to using a decoupled database that Elastic Beanstalk created, you can also create a database instance outside of your Elastic Beanstalk environment. Both options result in a database instance that's external to your Elastic Beanstalk environment and will require additional security group and connection string configuration. For more information, see Using Elastic Beanstalk with Amazon RDS.

Sections

Database lifecycle

You can choose what you want to happen to the database after you decouple it from your Elastic Beanstalk environment. The options that you can choose from are collectively referred to as *deletion policies*. The following deletion policies apply to a database after you decouple it from an Elastic Beanstalk environment or terminate the Elastic Beanstalk environment.

- *Snapshot* — Before Elastic Beanstalk terminates the database, it saves a snapshot of it. You can restore a database from a snapshot when you add a DB instance to an Elastic Beanstalk environment or when you create a standalone database. For more information about creating a new standalone DB instance from a snapshot, see Restoring from a DB snapshot in the *Amazon RDS User Guide*. You might incur charges for storing database snapshots. For more information, see the *Backup Storage* section of Amazon RDS Pricing.
- *Delete* — Elastic Beanstalk terminates the database. After it's terminated, the database instance is no longer available for any operation.
- *Retain* — The database instance isn't terminated. It remains available and operational, though decoupled from Elastic Beanstalk. You can then configure one or multiple environments to

connect to the database as an external Amazon RDS database instance. For more information, see Using Elastic Beanstalk with Amazon RDS.

Adding an Amazon RDS DB instance to your environment using the console

You can add a DB instance to your environment by using the Elastic Beanstalk console.

To add a DB instance to your environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Choose a DB engine, and enter a user name and password.
6. To save the changes choose **Apply** at the bottom of the page.

You can configure the following options:

- **Snapshot** – Choose an existing database snapshot. Elastic Beanstalk restores the snapshot and adds it to your environment. The default value is **None**. When the value is **None**, you can configure a new database using the other settings on this page.
- **Engine** – Choose a database engine.
- **Engine version** – Choose a specific version of the database engine.
- **Instance class** – Choose the DB instance class. For information about DB instance classes, see https://aws.amazon.com/rds/.
- **Storage** – Choose the amount of storage to provision for your database. You can increase allocated storage later, but you can't decrease it. For information about storage allocation, see Features.
- **Username** – Enter a user name of your choice using a combination of only numbers and letters.
- **Password** – Enter a password of your choice containing 8–16 printable ASCII characters (excluding /, \, and @).

- **Availability** – Choose **High (Multi-AZ)** to run a warm backup in a second Availability Zone for high availability.
- **Database deletion policy** – The deletion policy determines what happens to the database after it's decoupled from your environment. It can be set to the following values: `Create Snapshot`, `Retain`, or `Delete`. These values are described in Database lifecycle in this same topic.

Note

Elastic Beanstalk creates a master user for the database using the user name and password you provide. To learn more about the master user and its privileges, see Master User Account Privileges.

It takes about 10 minutes to add a DB instance. When the update is complete the new database is *coupled* to your environment. The hostname and other connection information for the DB instance are available to your application through the following environment properties.

Property name	Description	Property value
`RDS_HOSTNAME`	The hostname of the DB instance.	On the **Connectivity & security** tab on the Amazon RDS console: **Endpoint**.
`RDS_PORT`	The port where the DB instance accepts connectio ns. The default value varies among DB engines.	On the **Connectivity & security** tab on the Amazon RDS console: **Port**.
`RDS_DB_NAME`	The database name, **ebdb**.	On the **Configuration** tab on the Amazon RDS console: **DB Name**.
`RDS_USERNAME`	The username that you configured for your database.	On the **Configuration** tab on the Amazon RDS console: **Master username**.

Property name	Description	Property value
RDS_PASSWORD	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

Connecting to the database

Use the connectivity information to connect to your database from inside your application through environment variables. For more information about using Amazon RDS with your applications, see the following topics.

- Java SE – Connecting to a database (Java SE platforms)
- Java with Tomcat – Connecting to a database (Tomcat platforms)
- Node.js – Connecting to a database
- .NET – Connecting to a database
- PHP – Connecting to a database with a PDO or MySQLi
- Python – Connecting to a database
- Ruby – Connecting to a database

Configuring an integrated RDS DB instance using the console

You can view and modify configuration settings for your database instance in the **Database** section on the environment's **Configuration** page in the Elastic Beanstalk console.

To configure your environment's DB instance in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.

You can modify the **Instance class**, **Storage, Password**, **Availability**, and **Database deletion policy** settings after database creation. If you change the instance class, Elastic Beanstalk re-provisions the DB instance.

If you no longer need Elastic Beanstalk to associate the database to the environment, you can choose to decouple it by selecting **Decouple database**. It's important to understand the options and considerations involved with this operation. For more information, see the section called "Decoupling an RDS DB instance using the console".

Warning

Don't modify settings on the coupled database instance outside of the functionality that's provided by Elastic Beanstalk (for example, in the Amazon RDS console). If you do, your Amazon RDS DB configuration might be out of sync with your environment's definition. When you update or restart your environment, the settings specified in the environment override any settings you made outside of Elastic Beanstalk.
If you need to modify settings that Elastic Beanstalk doesn't directly support, use Elastic Beanstalk configuration files.

Configuring an integrated RDS DB instance using configuration files

You can configure your environment's database instance using configuration files. Use the options in the `aws:rds:dbinstance` namespace. The following example modifies the allocated database storage size to 100 GB.

Example .ebextensions/db-instance-options.config

```
option_settings:
  aws:rds:dbinstance:
    DBAllocatedStorage: 100
```

If you want to configure DB instance properties that Elastic Beanstalk doesn't support, you can still use a configuration file, and specify your settings using the `resources` key. The following example sets values to the `StorageType` and `Iops` Amazon RDS properties.

Example .ebextensions/db-instance-properties.config

```
Resources:
  AWSEBRDSDatabase:
    Type: AWS::RDS::DBInstance
    Properties:
      StorageType:io1
      Iops: 1000
```

Decoupling an RDS DB instance using the console

You can decouple your database from an Elastic Beanstalk environment without affecting the health of the environment. Consider the following requirements before you decouple the database:

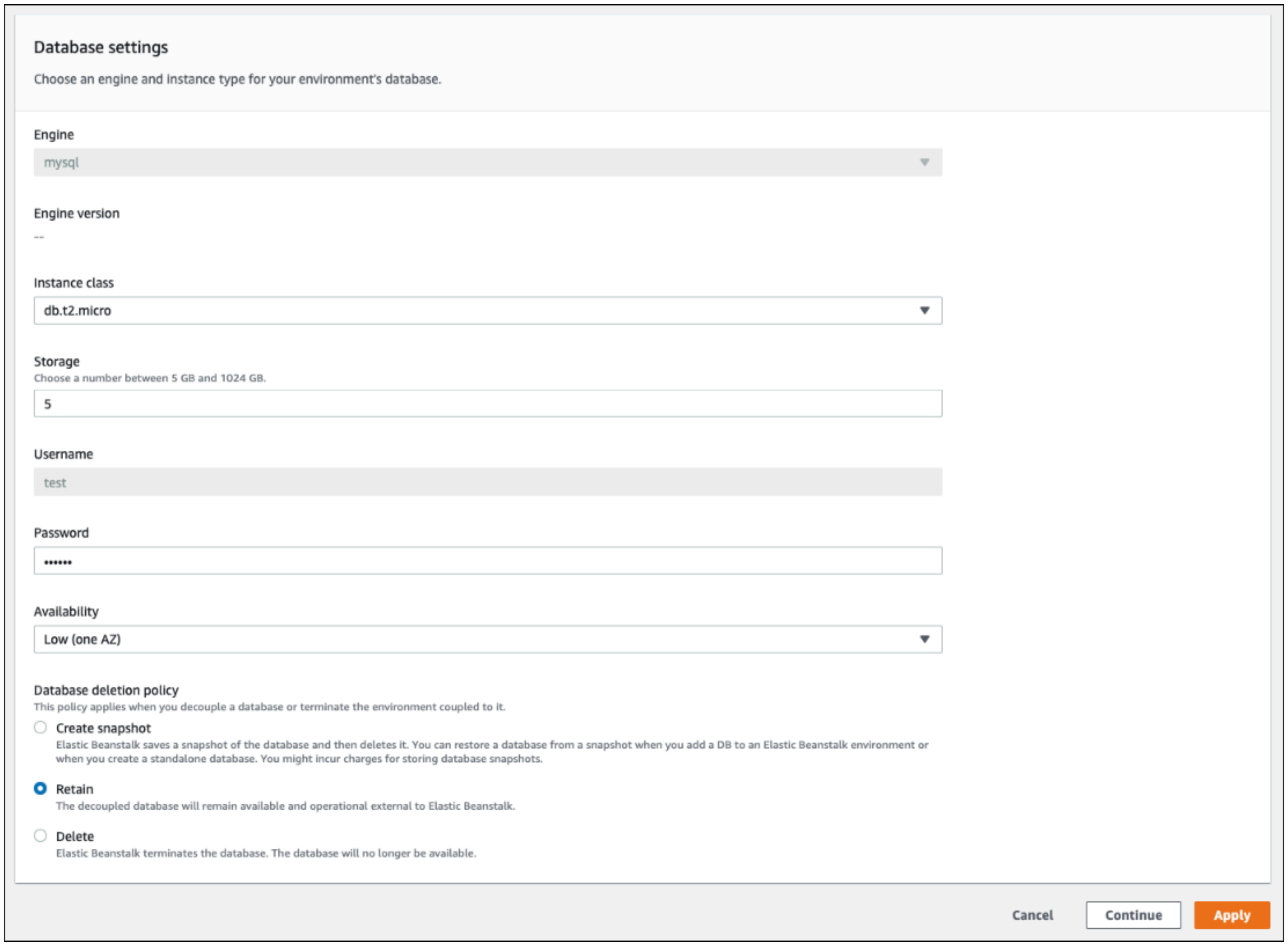
- *What should happen to the database after it's decoupled?*

  You can choose to create a snapshot of the database and then terminate it, retain the database operational as a standalone database external to Elastic Beanstalk, or permanently delete the database. The **Database deletion policy** setting determines this result. For a detailed description of the deletion policies, see Database lifecycle in this same topic.
- *Do you need make any changes to the database configuration settings before decoupling it?*

  If you need to make any configuration changes to the database, you should apply them *before decoupling* the database. This includes changes to the **Database deletion policy**. Any pending changes that are submitted simultaneously with the **Decouple database** setting will be ignored, while only the decouple setting is applied.

To decouple a DB instance from an environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Review all of the configurations values in the **Database settings** section, especially the **Database deletion policy**, which determines what happens to the database after it's decoupled.



Database settings

Choose an engine and instance type for your environment's database.

Engine: mysql

Engine version: --

Instance class: db.t2.micro

Storage
Choose a number between 5 GB and 1024 GB.
5

Username: test

Password: ••••••

Availability: Low (one AZ)

Database deletion policy
This policy applies when you decouple a database or terminate the environment coupled to it.

- ○ Create snapshot
  Elastic Beanstalk saves a snapshot of the database and then deletes it. You can restore a database from a snapshot when you add a DB to an Elastic Beanstalk environment or when you create a standalone database. You might incur charges for storing database snapshots.
- ● Retain
  The decoupled database will remain available and operational external to Elastic Beanstalk.
- ○ Delete
  Elastic Beanstalk terminates the database. The database will no longer be available.

If all of the other configuration settings are correct, skip to **Step 6** to decouple the database.

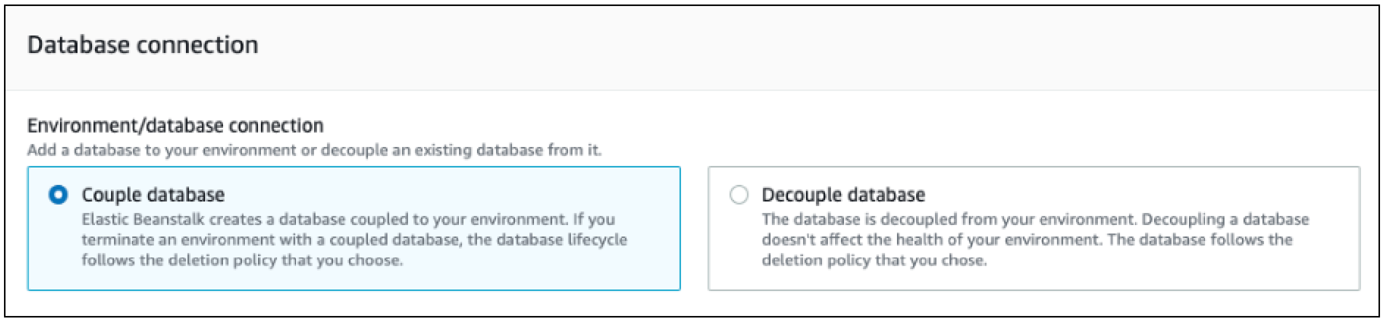
Warning

It's important to *apply* the **Database deletion policy** setting *separately* from **Decouple database**. If you select **Apply** with the intent to save both **Decouple database** and a newly selected **Database deletion policy**, the new deletion policy that you chose will be ignored. Elastic Beanstalk will decouple the database following the prior-set deletion policy. If the prior-set deletion policy is `Delete` or `Create Snapshot`, you risk losing the database instead of following the intended pending policy.

If any of the configuration settings require updates do the following:

   1. Make the required modifications in the **Database settings** panel.
   2. Choose **Apply**. It will take a few minutes to save the configuration changes for your database.
   3. Go back to **Step 3** and choose **Configuration** from the navigation pane.

6. Go to the **Database connection** section of the pane.


Database connection

Environment/database connection
Add a database to your environment or decouple an existing database from it.

[✓ Couple database]
Elastic Beanstalk creates a database coupled to your environment. If you terminate an environment with a coupled database, the database lifecycle follows the deletion policy that you choose.

Decouple database
The database is decoupled from your environment. Decoupling a database doesn't affect the health of your environment. The database follows the deletion policy that you chose.

7. Choose **Decouple database**.
8. Choose **Apply** to initiate the database decoupling operation.

The deletion policy setting determines the outcome for the database and the length of time that's required to decouple the database.

- If the deletion policy is set to `Delete`, the database is deleted. The operation can take approximately 10-20 minutes, depending on the size of database.
- If the deletion policy is set to `Snapshot`, a snapshot of the database is created. Then, the database is deleted. The length of time required for this process varies according to the size of the database.
- If the deletion policy is set to `Retain`, the database remains operational external to the Elastic Beanstalk environment. It usually takes less than five minutes to decouple a database.

If you decided to retain the database external to your Elastic Beanstalk environment, you'll need to take additional steps to configure it. For more information, see Using Elastic Beanstalk with Amazon RDS. If you plan to use the database that you decouple for a production environment, verify the storage type that the database uses is suitable for your workload. For more information, see DB Instance Storage and Modifying a DB instance in the *Amazon RDS User Guide*.

Decoupling an RDS DB instance using configuration files

You can decouple your DB instance from an Elastic Beanstalk environment without affecting the health of the environment. The database instance follows the *database deletion policy* that was applied when the database was decoupled.

Both of the options required to decouple the database are in the the section called "aws:rds:dbinstance" namespace. They are as follows:

- The `DBDeletionPolicy` option sets the deletion policy. It can be set to the following values: `Snapshot`, `Delete`, or `Retain`. These values are described in Database lifecycle in this same topic.
- The `HasCoupledDatabase` option determines if your environment has a coupled database.
  - If toggled to `true`, Elastic Beanstalk creates a new DB instance coupled to your environment.
  - If toggled to `false`, Elastic Beanstalk starts decoupling the DB instance from your environment.

If you want to change your database configuration before you decouple it, apply any configuration changes first, in a separate operation. This includes changing the `DBDeletionPolicy` configuration. After your changes are applied, run a separate command to set the decoupling option. If you submit other configuration settings and the decouple setting at the same time, the other configuration option settings are ignored while the decouple setting is applied.

Warning

It's important that you run the commands to apply the `DBDeletionPolicy` and `HasCoupledDatabase` settings as two separate operations. If the active deletion policy is already set to `Delete` or `Snapshot`, you risk losing the database. The database follows the deletion policy that's currently active, rather than the pending deletion policy that you intended.

To decouple a DB instance from an environment

Follow these steps to decouple the database from your Elastic Beanstalk environment. You can use the EB CLI or the AWS CLI to complete the steps. For more information, see Advanced environment customization with configuration files.

1. If you want to change the deletion policy, set up a configuration file in the following format. In this example, the deletion policy is set to retain.

   Example

   ```
   option_settings:
     aws:rds:dbinstance:
       DBDeletionPolicy: Retain
   ```

2. Run the command using your preferred tool to complete the configuration update.
3. Set up a configuration file to set `HasCoupledDatabase` to `false`.

   Example

   ```
   option_settings:
     aws:rds:dbinstance:
       HasCoupledDatabase: false
   ```

4. Run the command using your preferred tool to complete the configuration update.

The deletion policy setting determines the outcome for the database and the length of time that's required to decouple the database.

- If the deletion policy is set to `Delete`, the database is deleted. The operation can take approximately 10-20 minutes, depending on the size of database.
- If the deletion policy is set to `Snapshot`, a snapshot of the database is created. Then, the database is deleted. The length of time required for this process varies according to the size of the database.
- If the deletion policy is set to `Retain`, the database remains operational external to the Elastic Beanstalk environment. It usually takes less than five minutes to decouple a database.

If you decided to retain the database external to your Elastic Beanstalk environment, you'll need to take additional steps to configure it. For more information, see Using Elastic Beanstalk with Amazon RDS. If you plan to use the database that you decouple for a production environment, verify the storage type that the database uses is suitable for your workload. For more information, see DB Instance Storage and Modifying a DB instance in the *Amazon RDS User Guide*.

Your AWS Elastic Beanstalk environment security

Elastic Beanstalk provides several options that control the service access (security) of your environment and of the Amazon EC2 instances in it. This topic discusses the configuration of these options.

Sections

- Configuring your environment security
- Environment security configuration namespaces

Configuring your environment security

You can modify your Elastic Beanstalk environment security configuration in the Elastic Beanstalk console.

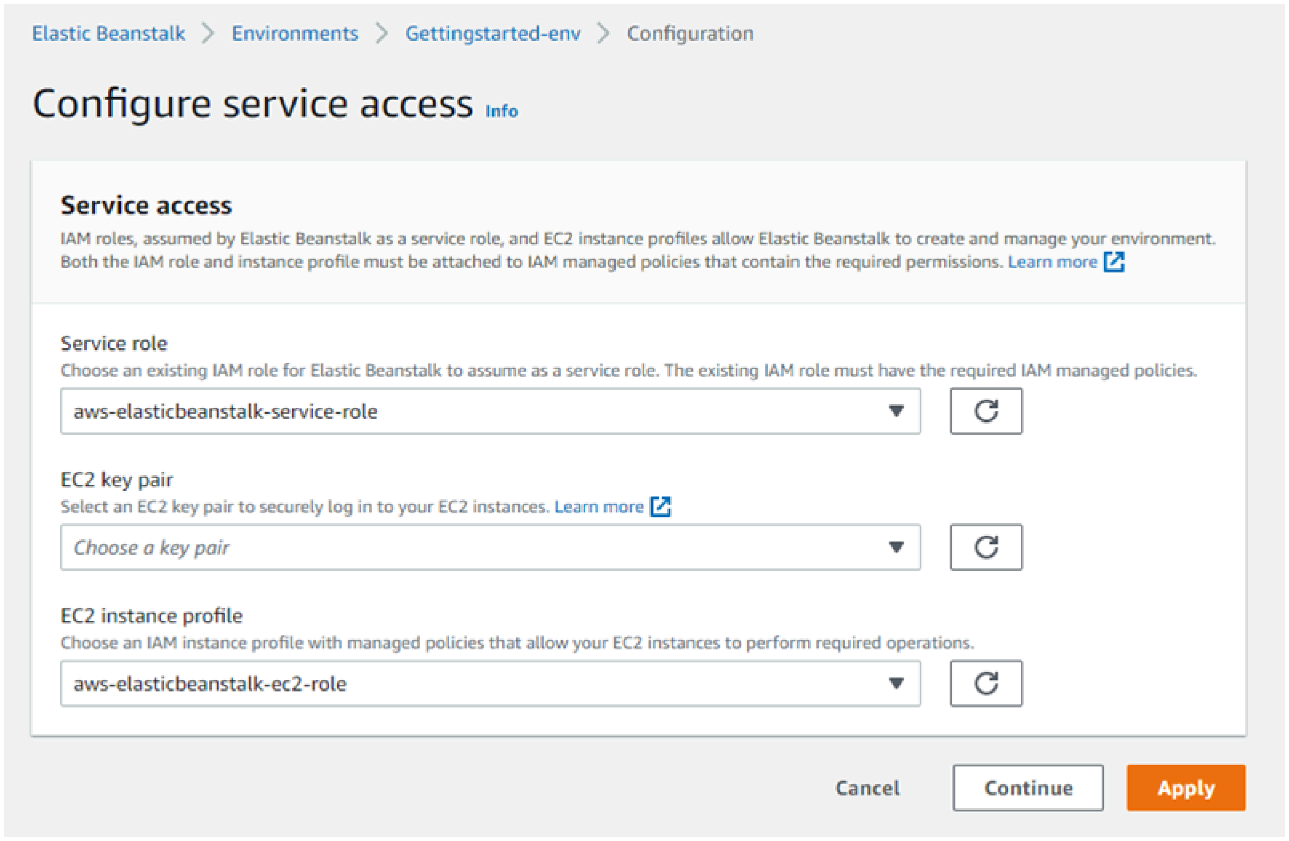
To configure environment service access (security) in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Service access** configuration category, choose **Edit**.

The following settings are available.

Settings

- Service role
- EC2 key pair
- IAM instance profile


Elastic Beanstalk > Environments > Gettingstarted-env > Configuration

Configure service access Info

Service access

IAM roles, assumed by Elastic Beanstalk as a service role, and EC2 instance profiles allow Elastic Beanstalk to create and manage your environment. Both the IAM role and instance profile must be attached to IAM managed policies that contain the required permissions. Learn more

Service role

Choose an existing IAM role for Elastic Beanstalk to assume as a service role. The existing IAM role must have the required IAM managed policies.

aws-elasticbeanstalk-service-role

EC2 key pair

Select an EC2 key pair to securely log in to your EC2 instances. Learn more

Choose a key pair

EC2 instance profile

Choose an IAM instance profile with managed policies that allow your EC2 instances to perform required operations.

aws-elasticbeanstalk-ec2-role

Service role

Select a service role to associate with your Elastic Beanstalk environment. Elastic Beanstalk assumes the service role when it accesses other AWS services on your behalf. For details, see Managing Elastic Beanstalk service roles.

EC2 key pair

You can securely log in to the Amazon Elastic Compute Cloud (Amazon EC2) instances provisioned for your Elastic Beanstalk application with an Amazon EC2 key pair. For instructions on creating a key pair, see Creating a Key Pair Using Amazon EC2 in the *Amazon EC2 User Guide*.

Note

When you create a key pair, Amazon EC2 stores a copy of your public key. If you no longer need to use it to connect to any environment instances, you can delete it from Amazon EC2. For details, see Deleting Your Key Pair in the *Amazon EC2 User Guide*.

Choose an **EC2 key pair** from the drop-down menu to assign it to your environment's instances. When you assign a key pair, the public key is stored on the instance to authenticate the private key, which you store locally. The private key is never stored on AWS.

For more information about connecting to Amazon EC2 instances, see Connect to Your Instance and Connecting to Linux/UNIX Instances from Windows using PuTTY in the *Amazon EC2 User Guide*.

IAM instance profile

An EC2 instance profile is an IAM role that is applied to instances launched in your Elastic Beanstalk environment. Amazon EC2 instances assume the instance profile role to sign requests to AWS and access APIs, for example, to upload logs to Amazon S3.

The first time you create an environment in the Elastic Beanstalk console, Elastic Beanstalk prompts you to create an instance profile with a default set of permissions. You can add permissions to this profile to provide your instances access to other AWS services. For details, see Managing Elastic Beanstalk instance profiles.

Note

Previously Elastic Beanstalk created a default EC2 instance profile named `aws-elasticbeanstalk-ec2-role` the first time an AWS account created an environment. This instance profile included default managed policies. If your account already has this instance profile, it will remain available for you to assign to your environments. However, recent AWS security guidelines don't allow an AWS service to automatically create roles with trust policies to other AWS services, EC2 in this case. Because of these security guidelines, Elastic Beanstalk no longer creates a default `aws-elasticbeanstalk-ec2-role` instance profile.

Note

There is another aspect of EC2 instance security that designates firewall rules for EC2 instances. This is controlled by EC2 security groups. For more information, see The Amazon EC2 instances for your Elastic Beanstalk environment.

Environment security configuration namespaces

Elastic Beanstalk provides configuration options in the following namespaces to enable you to customize the security of your environment:

- `aws:elasticbeanstalk:environment` – Configure the environment's service role using the `ServiceRole` option.
- `aws:autoscaling:launchconfiguration` – Configure permissions for the environment's Amazon EC2 instances using the `EC2KeyName`, `IamInstanceProfile`, `DisableDefaultEC2SecurityGroup`, and `SecurityGroups` options.

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See Recommended values for details.

Tagging resources in your Elastic Beanstalk environments

You can apply tags to your AWS Elastic Beanstalk environments. Tags are key-value pairs associated with AWS resources. For information about Elastic Beanstalk resource tagging, use cases, tag key and value constraints, and supported resource types, see Tagging Elastic Beanstalk application resources.

Elastic Beanstalk applies environment tags to the environment resource itself, as well as to other AWS resources that Elastic Beanstalk creates for the environment. You can use tags to manage permissions at the specific resource level within an environment. For more information, see Tagging Your Amazon EC2 Resources in the *Amazon EC2 User Guide*.

By default, Elastic Beanstalk applies a few tags to your environment:

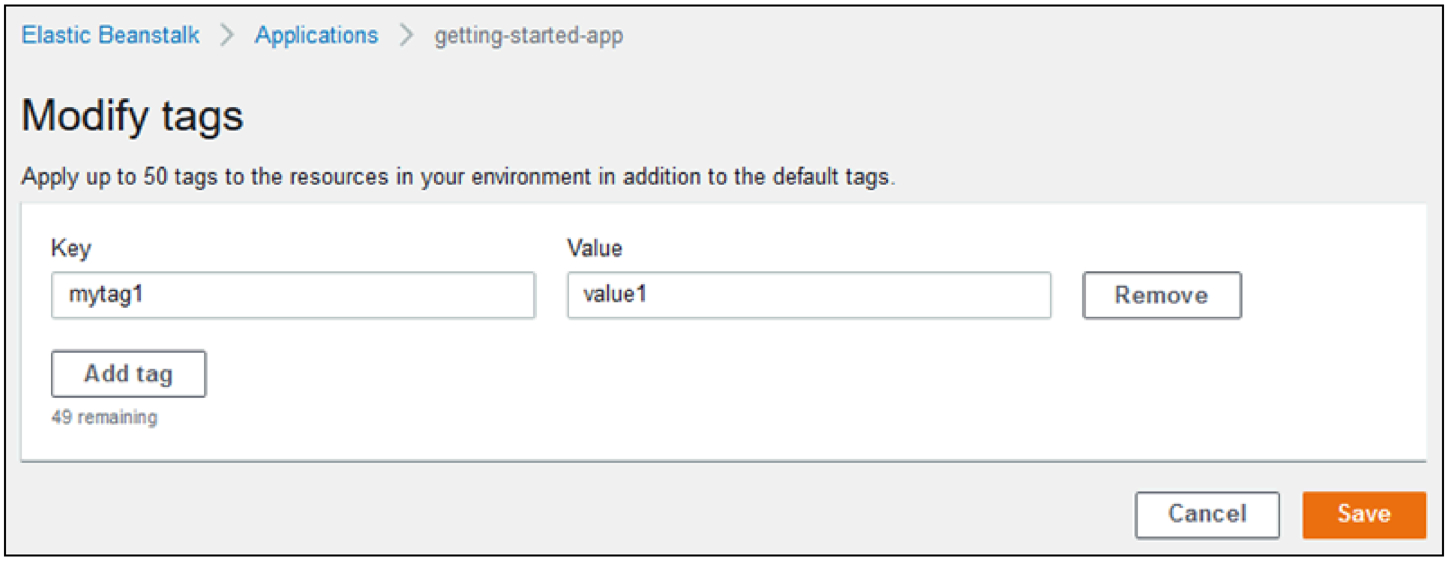
- `elasticbeanstalk:environment-name` – The name of the environment.
- `elasticbeanstalk:environment-id` – The environment ID.
- `Name` – Also the name of the environment. `Name` is used in the Amazon EC2 dashboard to identify and sort resources.

You can't edit these default tags.

You can specify tags when you create the Elastic Beanstalk environment. In an existing environment, you can add or remove tags, and update the values of existing tags. An environment can have up to 50 tags including the default tags.

Adding tags during environment creation

When you use the Elastic Beanstalk console to create an environment, you can specify tag keys and values on the **Modify tags** configuration page of the Create New Environment wizard.



Elastic Beanstalk > Applications > getting-started-app

Modify tags

Apply up to 50 tags to the resources in your environment in addition to the default tags.

Key	Value
mytag1	value1

If you use the EB CLI to create an environment, use the `--tags` option with **eb create** to add tags.

```
~/workspace/my-app$ eb create --tags mytag1=value1,mytag2=value2
```

With the AWS CLI or other API-based clients, use the `--tags` parameter on the **create-environment** command.

```
$ aws elasticbeanstalk create-environment \
      --tags Key=mytag1,Value=value1 Key=mytag2,Value=value2 \
      --application-name my-app --environment-name my-env --cname-prefix my-app --
version-label v1 --template-name my-saved-config
```

Saved configurations include user-defined tags. When you apply a saved configuration that contains tags during environment creation, those tags are applied to the new environment, as long as you don't specify any new tags. If you add tags to an environment using one of the preceding methods, any tags defined in the saved configuration are discarded.

Managing tags of an existing environment

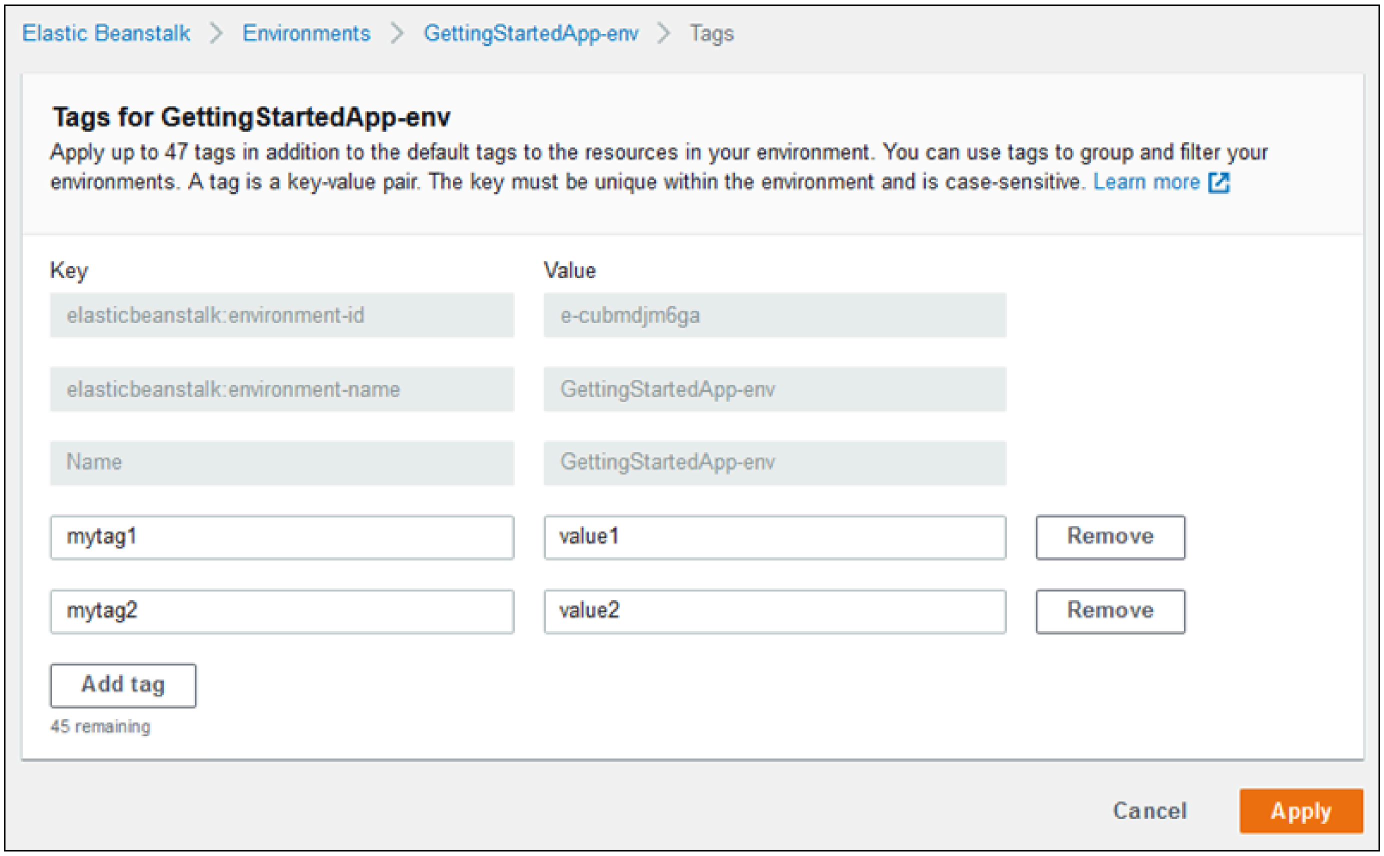
You can add, update, and delete tags in an existing Elastic Beanstalk environment. Elastic Beanstalk applies the changes to your environment's resources.

However, you can't edit the default tags that Elastic Beanstalk applies to your environment.

To manage an environment's tags in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Tags**.

   The tag management page shows the list of tags that currently exist in the environment.



Elastic Beanstalk > Environments > GettingStartedApp-env > Tags

Tags for GettingStartedApp-env

Apply up to 47 tags in addition to the default tags to the resources in your environment. You can use tags to group and filter your environments. A tag is a key-value pair. The key must be unique within the environment and is case-sensitive. Learn more

Key	Value
elasticbeanstalk:environment-id	e-cubmdjm6ga
elasticbeanstalk:environment-name	GettingStartedApp-env
Name	GettingStartedApp-env
mytag1	value1
mytag2	value2

4. Add, update, or delete tags:

   - To add a tag, enter it into the empty boxes at the bottom of the list. To add another tag, choose **Add tag** and Elastic Beanstalk adds another pair of empty boxes.

   - To update a tag's key or value, edit the respective box in the tag's row.
   - To delete a tag, choose **Remove** next to the tag's value box.
5. To save the changes choose **Apply** at the bottom of the page.

If you use the EB CLI to update your environment, use **eb tags** to add, update, delete, or list tags.

For example, the following command lists the tags in your default environment.

```
~/workspace/my-app$ eb tags --list
```

The following command updates the tag `mytag1` and deletes the tag `mytag2`.

```
~/workspace/my-app$ eb tags --update mytag1=newvalue --delete mytag2
```

For a complete list of options and more examples, see `eb tags`.

With the AWS CLI or other API-based clients, use the **list-tags-for-resource** command to list the tags of an environment.

```
$ aws elasticbeanstalk list-tags-for-resource --resource-arn
 "arn:aws:elasticbeanstalk:us-east-2:my-account-id:environment/my-app/my-env"
```

Use the **update-tags-for-resource** command to add, update, or delete tags in an environment.

```
$ aws elasticbeanstalk update-tags-for-resource \
      --tags-to-add Key=mytag1,Value=newvalue --tags-to-remove mytag2 \
      --resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-id:environment/my-
app/my-env"
```

Specify both tags to add and tags to update in the `--tags-to-add` parameter of **update-tags-for-resource**. A nonexisting tag is added, and an existing tag's value is updated.

Note

To use these two AWS CLI commands with an Elastic Beanstalk environment, you need the environment's ARN. You can retrieve the ARN by using the following command.

```
$ aws elasticbeanstalk describe-environments
```

Environment variables and other software settings

The **Configure updates, monitoring, and logging** configuration page lets you configure the software on the Amazon Elastic Compute Cloud (Amazon EC2) instances that run your application. You can configure environment variables, AWS X-Ray debugging, instance log storing and streaming, and platform-specific settings.

Topics

Configure platform-specific settings

In addition to the standard set of options available for all environments, most Elastic Beanstalk platforms let you specify language-specific or framework-specific settings. These appear in the **Platform software** section of the **Configure updates, monitoring, and logging** page, and can take the following forms.

- **Preset environment properties** – The Ruby platform uses environment properties for framework settings, such as `RACK_ENV` and `BUNDLE_WITHOUT`.
- **Placeholder environment properties** – The Tomcat platform defines an environment property named `JDBC_CONNECTION_STRING` that is not set to any value. This type of setting was more common on older platform versions.
- **Configuration options** – Most platforms define configuration options in platform-specific or shared namespaces, such as `aws:elasticbeanstalk:xray` or `aws:elasticbeanstalk:container:python`.

To configure platform-specific settings in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. Under **Platform software**, make necessary option setting changes.
6. To save the changes choose **Apply** at the bottom of the page.

For information about platform-specific options, and about getting environment property values in your code, see the platform topic for your language or framework:

- Docker – the section called “Environment configuration”
- Go – Using the Elastic Beanstalk Go platform
- Java SE – Using the Elastic Beanstalk Java SE platform
- Tomcat – Using the Elastic Beanstalk Tomcat platform
- .NET Core on Linux – Using the Elastic Beanstalk .NET core on Linux platform
- .NET – Using the Elastic Beanstalk .NET Windows platform
- Node.js – Using the Elastic Beanstalk Node.js platform
- PHP – Using the Elastic Beanstalk PHP platform
- Python – Using the Elastic Beanstalk Python platform
- Ruby – Using the Elastic Beanstalk Ruby platform

Configuring environment properties (environment variables)

You can use **environment properties**, (also known as **environment variables**), to pass endpoints, debug settings, and other information to your application. Environment variables help you run your application in multiple environments for different purposes, such as development, testing, staging, and production.

In addition, when you add a database to your environment, Elastic Beanstalk sets environment variables, such as `RDS_HOSTNAME`, that you can read in your application code to construct a connection object or string.

To configure environment variables in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. Scroll down to **Runtime environment variables**.
6. Select **Add environment variable**.
7. For **Source** select **Plain text**.

> **Note**
>
> The **Secrets Manager** and **SSM Parameter Store** values in the drop-down are for configuring environment variables as secrets to store sensitive data, such as credentials and API keys. For more information, see Using Elastic Beanstalk with AWS Secrets Manager and AWS Systems Manager Parameter Store.

8. Enter the **Environment variable name** and **Environment variable value** pairs.
9. If you need to add more variables repeat **Step 6** through **Step 8**.
10. To save the changes choose **Apply** at the bottom of the page.

Environment property limits

- **Keys** can contain any alphanumeric characters and the following symbols: `_ . : / + \ - @`

  The symbols listed are valid for environment property keys, but might not be valid for environment variable names on your environment's platform. For compatibility with all platforms, limit environment properties to the following pattern: `[A-Z_][A-Z0-9_]*`
- **Values** can contain any alphanumeric characters, white space, and the following symbols: `_ . : / = + \ - @ ' "`

> **Note**
>
> Some characters in environment property values must be escaped. Use the backslash character (\) to represent some special characters and control characters. The following list includes examples for representing some characters that need to be escaped:
>
> - backslash (\) — to represent use \\
> - single quote (') — to represent use \'

  - double quote (") — to represent use \"

- **Keys** and **values** are case sensitive.
- The combined size of all environment properties cannot exceed 4,096 bytes when stored as strings with the format *key*=*value*.

Software setting namespaces

You can use a configuration file to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be platform specific or apply to all platforms in the Elastic Beanstalk service as a whole. Configuration options are organized into *namespaces*.

You can use Elastic Beanstalk configuration files to set environment properties and configuration options in your source code. Use the `aws:elasticbeanstalk:application:environment` namespace to define environment properties.

Example .ebextensions/options.config

```
option_settings:
  aws:elasticbeanstalk:application:environment:
    API_ENDPOINT: www.example.com/api
```

If you use configuration files or AWS CloudFormation templates to create custom resources, you can use an AWS CloudFormation function to get information about the resource and assign it to an environment property dynamically during deployment. The following example from the elastic-beanstalk-samples GitHub repository uses the Ref function to get the ARN of an Amazon SNS topic that it creates, and assigns it to an environment property named `NOTIFICATION_TOPIC`.

Notes

- If you use an AWS CloudFormation function to define an environment property, the Elastic Beanstalk console displays the value of the property before the function is evaluated. You can use the `get-config` platform script to confirm the values of environment properties that are available to your application.

- The Multicontainer Docker platform doesn't use AWS CloudFormation to create container resources. As a result, this platform doesn't support defining environment properties using AWS CloudFormation functions.

Example .Ebextensions/sns-topic.config

```
Resources:
  NotificationTopic:
    Type: AWS::SNS::Topic

option_settings:
  aws:elasticbeanstalk:application:environment:
    NOTIFICATION_TOPIC: '`{"Ref" : "NotificationTopic"}`'
```

You can also use this feature to propagate information from AWS CloudFormation pseudo parameters. This example gets the current region and assigns it to a property named `AWS_REGION`.

Example .Ebextensions/env-regionname.config

```
option_settings:
  aws:elasticbeanstalk:application:environment:
    AWS_REGION: '`{"Ref" : "AWS::Region"}`'
```

Most Elastic Beanstalk platforms define additional namespaces with options for configuring software that runs on the instance, such as the reverse proxy that relays requests to your application. For more information about the namespaces available for your platform, see the following:

- Go – Go configuration namespace
- Java SE – Java SE configuration namespace
- Tomcat – Tomcat configuration namespaces
- .NET Core on Linux – .NET Core on Linux configuration namespace
- .NET – The aws:elasticbeanstalk:container:dotnet:apppool namespace
- Node.js – Node.js configuration namespace
- PHP – Namespaces for configuration
- Python – Python configuration namespaces

- Ruby – Ruby configuration namespaces

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See Configuration options for more information.

Accessing environment properties

In most cases, you access environment properties in your application code like an environment variable. In general, however, environment properties are passed only to the application and can't be viewed by connecting an instance in your environment and running `env`.

- Go – `os.Getenv`

```
endpoint := os.Getenv("API_ENDPOINT")
```

- Java SE – `System.getenv`

```
String endpoint = System.getenv("API_ENDPOINT");
```

- Tomcat – `System.getProperty` and `System.getenv`

  Tomcat platform versions released on or after March 26, 2025, can also use `System.getenv` to access plaintext environment variables. You can continue to use `System.getProperty` to access plaintext environment variables. However, environment variables stored as secrets are only available using `System.getenv`.

```
String endpoint = System.getProperty("API_ENDPOINT");
```

```
String endpoint = System.getenv("API_ENDPOINT");
```

Important

The addition of `System.getenv` access for environment variables in Tomcat platform versions released on or after March 26, 2025 may cause unexpected behavior in applications that give environment variables precedence over Java system properties or when explicitly switching from System.getProperty to System.getenv. For more information and recommended actions, see Using the Elastic Beanstalk Tomcat platform.

- .NET Core on Linux – Environment.GetEnvironmentVariable

```
string endpoint = Environment.GetEnvironmentVariable("API_ENDPOINT");
```

- .NET – appConfig

```
NameValueCollection appConfig = ConfigurationManager.AppSettings;
string endpoint = appConfig["API_ENDPOINT"];
```

- Node.js – process.env

```
var endpoint = process.env.API_ENDPOINT
```

- PHP – $_SERVER

```
$endpoint = $_SERVER['API_ENDPOINT'];
```

- Python – os.environ

```
import os
endpoint = os.environ['API_ENDPOINT']
```

- Ruby – ENV

```
endpoint = ENV['API_ENDPOINT']
```

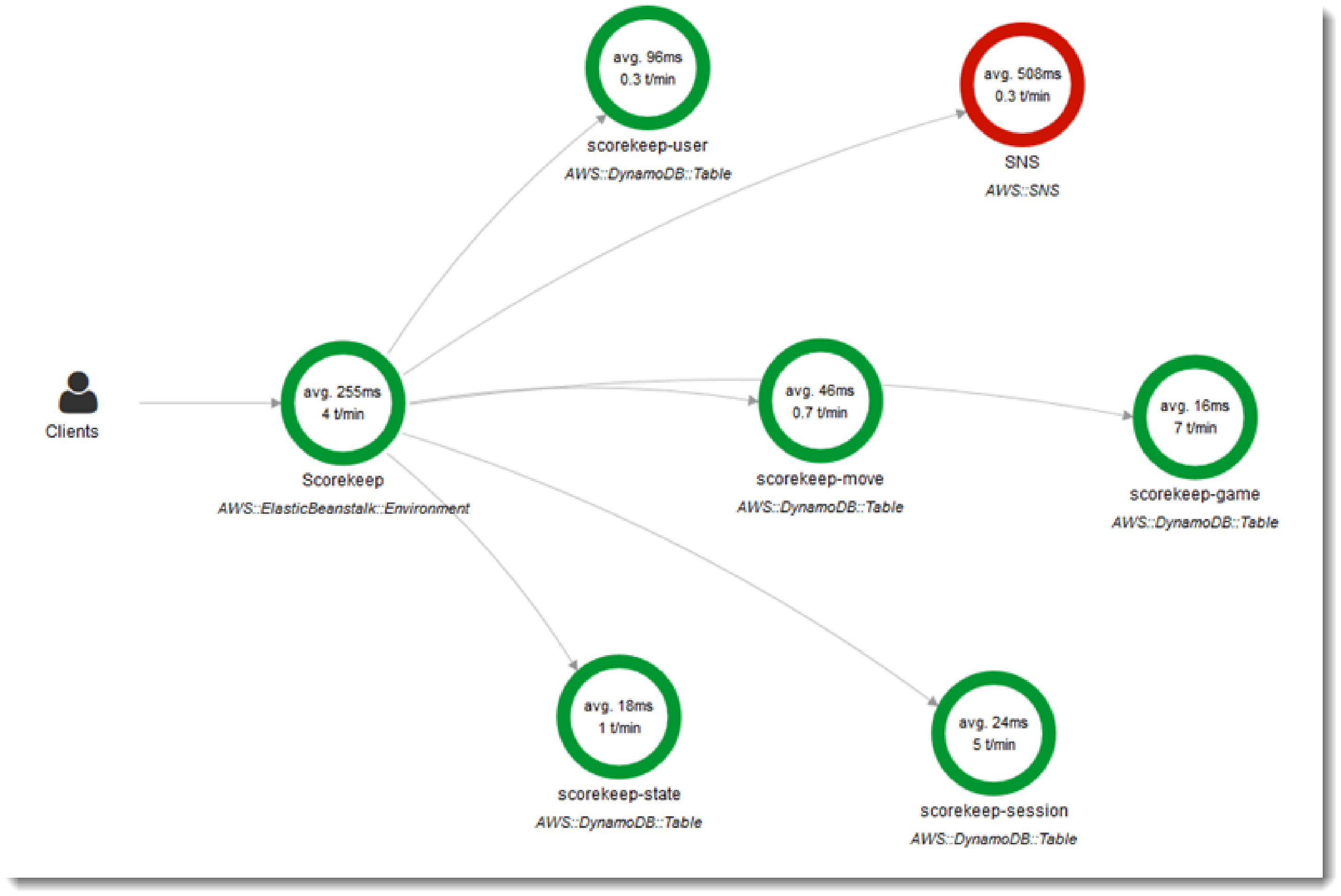
Outside of application code, such as in a script that runs during deployment, you can access environment properties with the get-config platform script. See the elastic-beanstalk-samples GitHub repository for example configurations that use get-config.

Configuring AWS X-Ray debugging

You can use the AWS Elastic Beanstalk console or a configuration file to run the AWS X-Ray daemon on the instances in your environment. X-Ray is an AWS service that gathers data about the requests that your application serves, and uses it to construct a service map that you can use to identify issues with your application and opportunities for optimization.

Note

Some regions don't offer X-Ray. If you create an environment in one of these regions, you can't run the X-Ray daemon on the instances in your environment.
For information about the AWS services offered in each Region, see Region Table.



X-Ray provides an SDK that you can use to instrument your application code, and a daemon application that relays debugging information from the SDK to the X-Ray API.

Supported platforms

You can use the X-Ray SDK with the following Elastic Beanstalk platforms:

- **Go** - version 2.9.1 and later
- **Java 8** - version 2.3.0 and later

- **Java 8 with Tomcat 8** - version 2.4.0 and later
- **Node.js** - version 3.2.0 and later
- **Windows Server** - all platform versions released on or after December 18th, 2016
- **Python** - version 2.5.0 and later

On supported platforms, you can use a configuration option to run the X-Ray daemon on the instances in your environment. You can enable the daemon in the Elastic Beanstalk console or by using a configuration file.

To upload data to X-Ray, the X-Ray daemon requires IAM permissions in the **AWSXrayWriteOnlyAccess** managed policy. These permissions are included in the Elastic Beanstalk instance profile. If you don't use the default instance profile, see Giving the Daemon Permission to Send Data to X-Ray in the *AWS X-Ray Developer Guide*.

Debugging with X-Ray requires the use of the X-Ray SDK. See the Getting Started with AWS X-Ray in the *AWS X-Ray Developer Guide* for instructions and sample applications.

If you use a platform version that doesn't include the daemon, you can still run it with a script in a configuration file. For more information, see Downloading and Running the X-Ray Daemon Manually (Advanced) in the *AWS X-Ray Developer Guide*.

Sections

Configuring debugging

You can enable the X-Ray daemon on a running environment in the Elastic Beanstalk console.

To enable debugging in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. In the **Amazon X-Ray** section, select **Activated**.

6. To save the changes choose **Apply** at the bottom of the page.

You can also enable this option during environment creation. For more information, see The create new environment wizard.

The aws:elasticbeanstalk:xray namespace

You can use the XRayEnabled option in the aws:elasticbeanstalk:xray namespace to enable debugging.

To enable debugging automatically when you deploy your application, set the option in a configuration file in your source code, as follows.

Example .ebextensions/debugging.config

```
option_settings:
  aws:elasticbeanstalk:xray:
    XRayEnabled: true
```

Viewing your Elastic Beanstalk environment logs

AWS Elastic Beanstalk provides two ways to regularly view logs from the Amazon EC2 instances that run your application:

- Configure your Elastic Beanstalk environment to upload rotated instance logs to the environment's Amazon S3 bucket.
- Configure the environment to stream instance logs to Amazon CloudWatch Logs.

When you configure instance log streaming to CloudWatch Logs, Elastic Beanstalk creates CloudWatch Logs log groups for proxy and deployment logs on the Amazon EC2 instances, and transfers these log files to CloudWatch Logs in real time. For more information about instance logs, see Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment.

In addition to instance logs, if you enable enhanced health for your environment, you can configure the environment to stream health information to CloudWatch Logs. When the environment's health status changes, Elastic Beanstalk adds a record to a health log group, with the new status and a description of the cause of the change. For information about environment health streaming, see Streaming Elastic Beanstalk environment health information to Amazon CloudWatch Logs.

Configuring instance log viewing

To view instance logs, you can enable instance log rotation and log streaming in the Elastic Beanstalk console.

To configure instance log rotation and log streaming in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. In the **S3 log storage** section, select **Activated** beneath **Rotate logs** to enable uploading rotated logs to Amazon S3.
6. in the **Instance log streaming to CloudWatch Logs** section, configure the following settings:
   - **Log streaming** – Select **Activated** to enable log streaming.
   - **Retention** – Specify the number of days to retain logs in CloudWatch Logs.
   - **Lifecycle** – Set to **Delete logs upon termination** to delete logs from CloudWatch Logs immediately if the environment is terminated, instead of waiting for them to expire.
7. To save the changes choose **Apply** at the bottom of the page.

After you enable log streaming, you can return to the **Software** configuration category or page and find the **Log Groups** link. Click this link to see your instance logs in the CloudWatch console.

Configuring environment health log viewing

To view environment health logs, you can enable environment health log streaming in the Elastic Beanstalk console.

To configure environment health log streaming in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.

4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. Go to the **Monitoring** section.
6. Under **Health event streaming to CloudWatch Logs**, configure the following settings:
   - **Log streaming** – Choose to **Activated** to enable log streaming.
   - **Retention** – Specify the number of days to retain logs in CloudWatch Logs.
   - **Lifecycle** – Set to **Delete logs upon termination** to delete logs from CloudWatch Logs immediately if the environment is terminated, instead of waiting for them to expire.
7. To save the changes choose **Apply** at the bottom of the page.

Log viewing namespaces

The following namespaces contain settings for log viewing:

- `aws:elasticbeanstalk:hostmanager` – Configure uploading rotated logs to Amazon S3.
- `aws:elasticbeanstalk:cloudwatch:logs` – Configure instance log streaming to CloudWatch.
- `aws:elasticbeanstalk:cloudwatch:logs:health` – Configure environment health streaming to CloudWatch.

Elastic Beanstalk environment notifications with Amazon SNS

You can configure your AWS Elastic Beanstalk environment to use Amazon Simple Notification Service (Amazon SNS) to notify you of important events that affect your application. To receive emails from AWS whenever an error occurs or the health of your environment changes, specify an email address when you create an environment or later on.

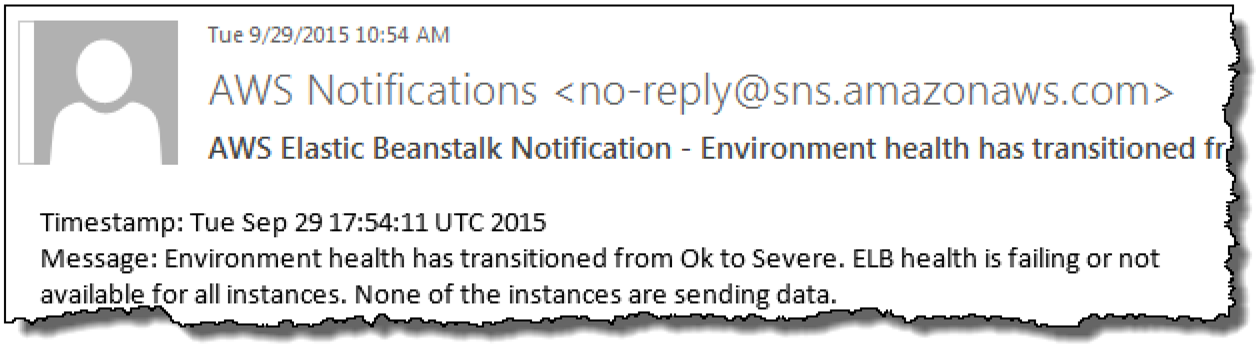
Note

Elastic Beanstalk uses Amazon SNS for notifications. For information about Amazon SNS pricing, see https://aws.amazon.com/sns/pricing/.

When you configure notifications for your environment, Elastic Beanstalk creates an Amazon SNS topic for your environment on your behalf. To send messages to an Amazon SNS topic, Elastic

Beanstalk must have the required permission. For more information, see Configuring permissions to send notifications.

When a notable event occurs, Elastic Beanstalk sends a message to the topic. Then, Amazon SNS relays the messages that it receives to the topic's subscribers. Notable events include environment creation errors and all changes in environment and instance health. Events for Amazon EC2 Auto Scaling operations (like adding and removing instances from the environment) and other informational events don't trigger notifications.



Tue 9/29/2015 10:54 AM

AWS Notifications <no-reply@sns.amazonaws.com>

AWS Elastic Beanstalk Notification - Environment health has transitioned fr

Timestamp: Tue Sep 29 17:54:11 UTC 2015
Message: Environment health has transitioned from Ok to Severe. ELB health is failing or not available for all instances. None of the instances are sending data.

You can enter an email address in the Elastic Beanstalk console when you create an environment or sometime afterwards. This will create an Amazon SNS topic and subscribe to it. Elastic Beanstalk manages the lifecycle of the topic, and deletes it when your environment is terminated or when you remove your email address in the environment management console.

The `aws:elasticbeanstalk:sns:topics` namespace provides options for configuring an Amazon SNS topic by using configuration files, a CLI, or an SDK. By using one of these methods, you can configure the type of subscriber and the endpoint. For type of subscriber, you can choose an Amazon SQS queue or HTTP URL.

You can only turn Amazon SNS notifications on or off. The frequency of notifications sent to the topic can be high, depending on the size and composition of your environment. For configuring notifications to be sent on specific circumstances, you have other options. You can set up event-driven rules with Amazon EventBridge that notify you when Elastic Beanstalk emits events that meet specific criteria. Or, alternatively, you can configure your environment to publish custom metrics and set Amazon CloudWatch alarms to notify you when those metrics reach an unhealthy threshold.

Configuring notifications using the Elastic Beanstalk console

You can enter an email address in the Elastic Beanstalk console to create an Amazon SNS topic for your environment.

To configure notifications in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. Scroll down to the **Email notifications** section.
6. Enter an email address.
7. To save the changes choose **Apply** at the bottom of the page.

When you enter an email address for notifications, Elastic Beanstalk creates an Amazon SNS topic for your environment and adds a subscription. Amazon SNS sends an email to the subscribed address to confirm the subscription. You must click the link in the confirmation email to activate the subscription and receive notifications.

Configuring notifications using configuration options

Use the options in the `aws:elasticbeanstalk:sns:topics` namespace to configure Amazon SNS notifications for your environment. You can set these options by using configuration files, a CLI, or an SDK.

- **Notification Endpoint** – The email address, Amazon SQS queue, or URL to send notifications to. If you set this option, then an SQS queue and a subscription for the specified endpoint are created. If the endpoint isn't an email address, you must also set the `Notification Protocol` option. SNS validates the value of `Notification Endpoint` based on the value of `Notification Protocol`. Setting this option multiple times creates additional subscriptions to the topic. If you remove this option, the topic is deleted.
- **Notification Protocol** – The protocol that's used to send notifications to the `Notification Endpoint`. This option defaults to `email`. Set this option to `email-json` to send JSON-formatted emails, `http` or `https` to post JSON-formatted notifications to an HTTP endpoint, or `sqs` to send notifications to an SQS queue.

  Note

  AWS Lambda notifications aren't supported.

- **Notification Topic ARN** – After setting a notification endpoint for your environment, read this setting to get the ARN of the SNS topic. You can also set this option to use an existing SNS topic for notifications. A topic that you attach to your environment though this option isn't deleted when you change this option or terminate the environment.

  To configure Amazon SNS notifications, you need to have the required permissions. If your IAM user uses the Elastic Beanstalk **AdministratorAccess-AWSElasticBeanstalk** managed user policy, then you should already have the required permissions to configure the default Amazon SNS topic that Elastic Beanstalk creates for your environment. However, if you configure an Amazon SNS topic that Elastic Beanstalk doesn't manage, then you need to add the following policy to your user role.

  JSON

  ```
  {
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "sns:SetTopicAttributes",
          "sns:GetTopicAttributes",
          "sns:Subscribe",
          "sns:Unsubscribe",
          "sns:Publish"
        ],
        "Resource": [
          "arn:aws:sns:us-east-2:123456789012:sns_topic_name"
        ]
      }
    ]
  }
  ```

- **Notification Topic Name** – Set this option to customize the name of the Amazon SNS topic used for environment notifications. If a topic with the same name already exists, Elastic Beanstalk attaches that topic to the environment.

> **Warning**
>
> If you attach an existing SNS topic to an environment with `Notification Topic Name`, Elastic Beanstalk will delete the topic in the event that you terminate the environment or change this setting sometime in the future.

If you change this option, the `Notification Topic ARN` is also changed. If a topic is already attached to the environment, Elastic Beanstalk deletes the old topic and creates a new topic and subscription.

By using a custom topic name, you must also provide an ARN of an externally created custom topic. The managed user policy doesn't automatically detect a topic with a custom name, so you must provide custom Amazon SNS permissions to your IAM users. Use a policy similar to the one that's used for a custom topic ARN, but include the following additions:

- Include two more actions in the `Actions` list, specifically: `sns:CreateTopic`, `sns:DeleteTopic`
- If you're changing the `Notification Topic Name` from one custom topic name to another, you must also include the ARNs of both topics in the `Resource` list. Alternatively, include a regular expression that covers both topics. This way Elastic Beanstalk has permissions to delete the old topic and create the new one.

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See Recommended values for details.

Configuring permissions to send notifications

This section discusses security considerations that are related to notifications that use Amazon SNS. There are two distinct cases:

- Use the default Amazon SNS topic that Elastic Beanstalk creates for your environment.
- Provide an external Amazon SNS topic through configuration options.

The default access policy for an Amazon SNS topic allows only the topic owner to publish or subscribe to it. However, through the proper policy configuration, Elastic Beanstalk can be granted permission to publish to an Amazon SNS topic in either one of the two cases described in this section. The following subsections provide more information.

Permissions for a default topic

When you configure notifications for your environment, Elastic Beanstalk creates an Amazon SNS topic for your environment. To send messages to an Amazon SNS topic, Elastic Beanstalk must have the required permission. If your environment uses the service role that the Elastic Beanstalk console or the EB CLI generated for it, or your account's monitoring service-linked role, then you don't need to do anything else. These managed roles include the necessary permission that allows Elastic Beanstalk to send messages to the Amazon SNS topic.

However, if you provided a custom service role when you created your environment, make sure that this custom service role includes the following policy.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sns:Publish"
      ],
      "Resource": [
        "arn:aws:sns:us-east-2:123456789012:ElasticBeanstalkNotifications*"
      ]
    }
  ]
}
```

Permissions for an external topic

Configuring notifications using configuration options explains how you can replace the Amazon SNS topic that Elastic Beanstalk provides with another Amazon SNS topic. If you replaced the topic, Elastic Beanstalk must verify that you have permission to publish to this SNS topic for you to be able to associate the SNS topic with the environment. You should have `sns:Publish`. The service role uses the same permission. To verify that this is the case, Elastic Beanstalk sends a test notification to SNS as part of your action to create or update the environment. If this test fails, then your attempt to create or update the environment also fails. Elastic Beanstalk displays a message that explains the reason for this failure.

If you provide a custom service role for your environment, make sure that your custom service role includes the following policy to allow Elastic Beanstalk to send messages to the Amazon SNS topic. In the following code, replace *sns_topic_name* with the name of the Amazon SNS topic that you provided in the configuration options.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sns:Publish"
      ],
      "Resource": [
        "arn:aws:sns:us-east-2:123456789012:sns_topic_name"
      ]
    }
  ]
}
```

For more information about Amazon SNS access control, see Example cases for Amazon SNS access control in the *Amazon Simple Notification Service Developer Guide*.

Configuring Amazon Virtual Private Cloud (Amazon VPC) with Elastic Beanstalk

Amazon Virtual Private Cloud (Amazon VPC) is the networking service that routes traffic securely to the EC2 instances that run your application in Elastic Beanstalk. If you don't configure a VPC when you launch your environment, Elastic Beanstalk uses the default VPC.

You can launch your environment in a custom VPC to customize networking and security settings. Elastic Beanstalk lets you choose which subnets to use for your resources, and how to configure

IP addresses for the instances and load balancer in your environment. An environment is locked to a VPC when you create it, but you can change subnet and IP address settings on a running environment.

Configuring VPC settings in the Elastic Beanstalk console

If you chose a custom VPC when you created your environment, you can modify its VPC settings in the Elastic Beanstalk console.

To configure your environment's VPC settings

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Network** configuration category, choose **Edit**.

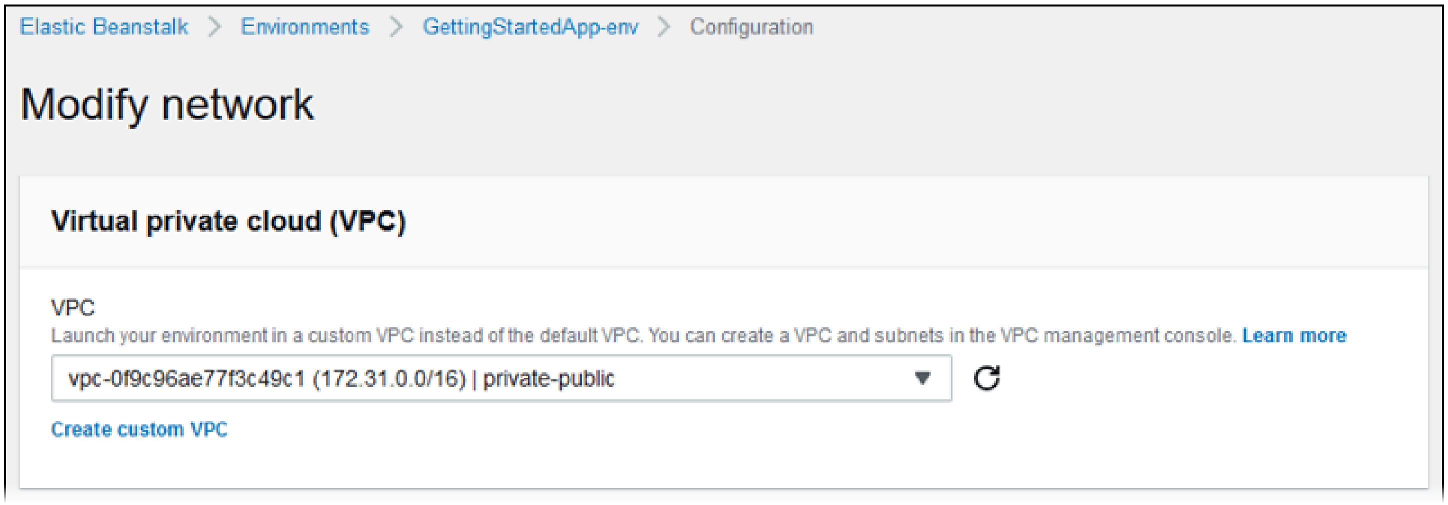
The following settings are available.

Options

- VPC
- Load balancer visibility
- Load balancer subnets
- Instance public IP address
- Instance subnets
- Database subnets

VPC

Choose a VPC for your environment. You can only change this setting during environment creation.


Elastic Beanstalk > Environments > GettingStartedApp-env > Configuration

Modify network

Virtual private cloud (VPC)

VPC

Launch your environment in a custom VPC instead of the default VPC. You can create a VPC and subnets in the VPC management console. Learn more

vpc-0f9c96ae77f3c49c1 (172.31.0.0/16) | private-public

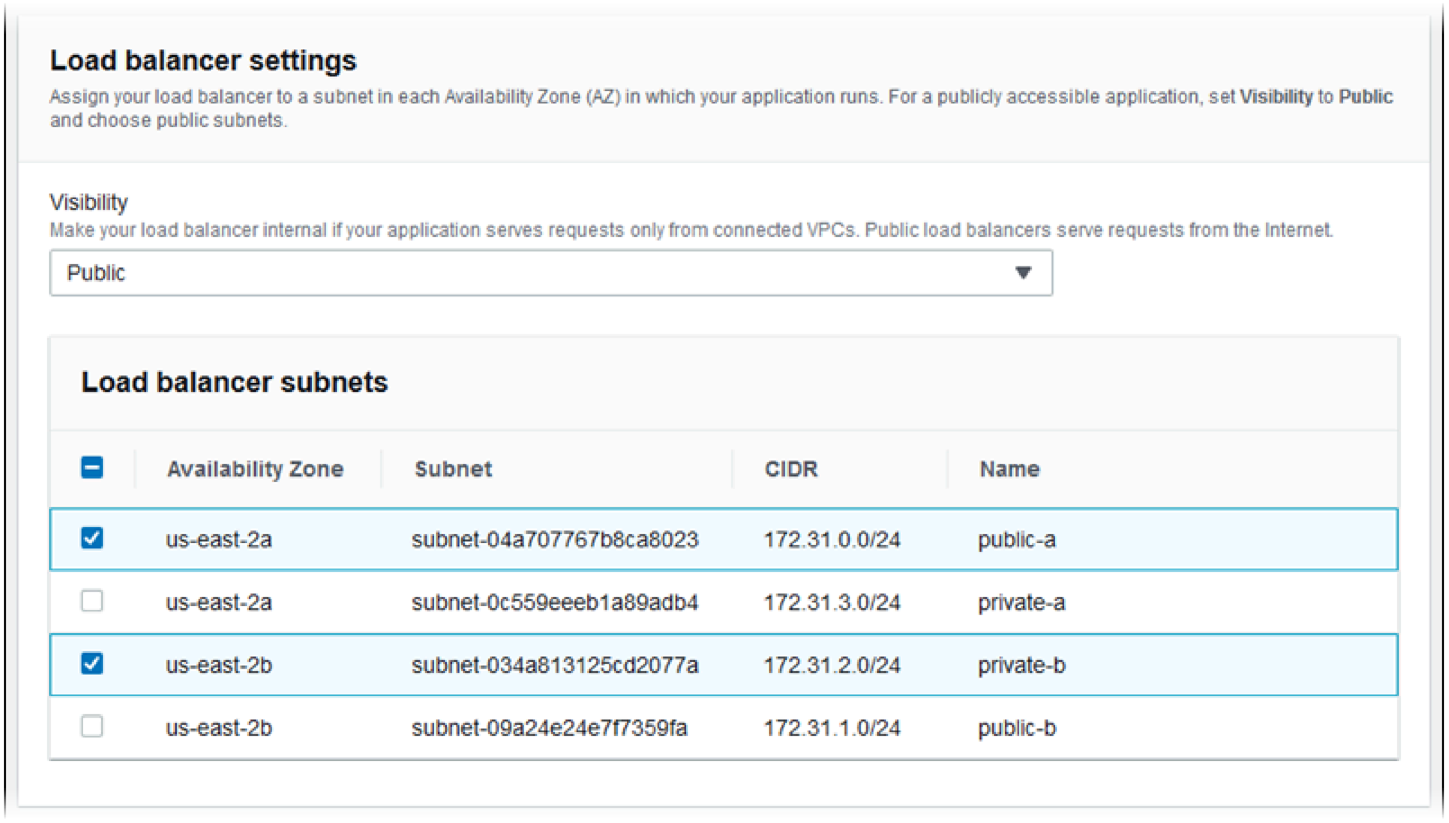
Create custom VPC

Load balancer visibility

For a load-balanced environment, choose the load balancer scheme. By default, the load balancer is public, with a public IP address and domain name. If your application only serves traffic from within your VPC or a connected VPN, deselect this option and choose private subnets for your load balancer to make the load balancer internal and disable access from the Internet.

Load balancer subnets

For a load-balanced environment, choose the subnets that your load balancer uses to serve traffic. For a public application, choose public subnets. Use subnets in multiple availability zones for high availability. For an internal application, choose private subnets and disable load balancer visibility.



Load balancer settings

Assign your load balancer to a subnet in each Availability Zone (AZ) in which your application runs. For a publicly accessible application, set **Visibility** to **Public** and choose public subnets.

Visibility

Make your load balancer internal if your application serves requests only from connected VPCs. Public load balancers serve requests from the Internet.

Public

Load balancer subnets

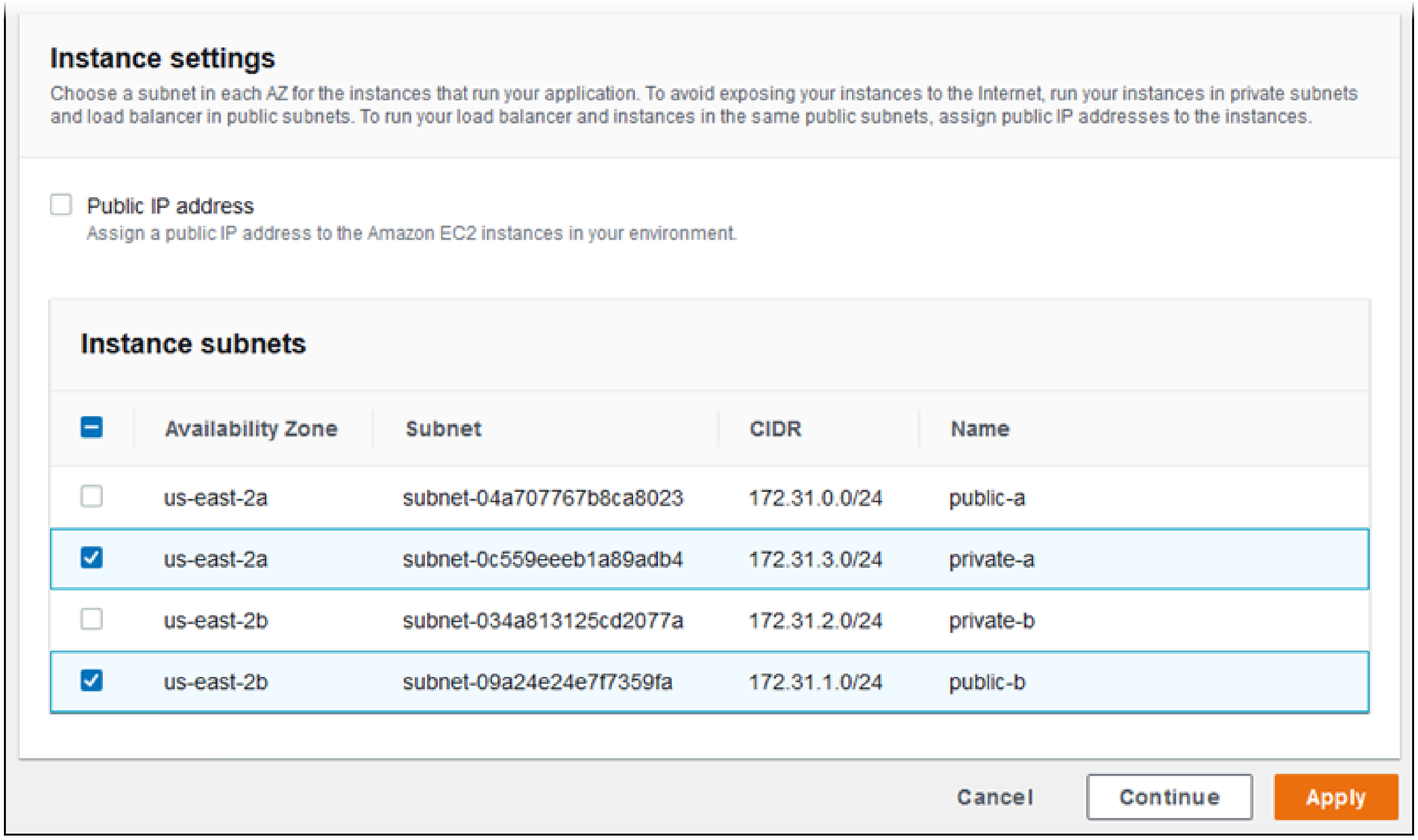
	Availability Zone	Subnet	CIDR	Name
✓	us-east-2a	subnet-04a707767b8ca8023	172.31.0.0/24	public-a
☐	us-east-2a	subnet-0c559eeeb1a89adb4	172.31.3.0/24	private-a
✓	us-east-2b	subnet-034a813125cd2077a	172.31.2.0/24	private-b
☐	us-east-2b	subnet-09a24e24e7f7359fa	172.31.1.0/24	public-b

Instance public IP address

If you choose public subnets for your application instances, enable public IP addresses to make them routable from the Internet.

Instance subnets

Choose subnets for your application instances. Choose at least one subnet for each availability zone that your load balancer uses. If you choose private subnets for your instances, your VPC must have a NAT gateway in a public subnet that the instances can use to access the Internet.



Instance settings

Choose a subnet in each AZ for the instances that run your application. To avoid exposing your instances to the Internet, run your instances in private subnets and load balancer in public subnets. To run your load balancer and instances in the same public subnets, assign public IP addresses to the instances.

☐ Public IP address
Assign a public IP address to the Amazon EC2 instances in your environment.

Instance subnets

	Availability Zone	Subnet	CIDR	Name
☐	us-east-2a	subnet-04a707767b8ca8023	172.31.0.0/24	public-a
✓	us-east-2a	subnet-0c559eeeb1a89adb4	172.31.3.0/24	private-a
☐	us-east-2b	subnet-034a813125cd2077a	172.31.2.0/24	private-b
✓	us-east-2b	subnet-09a24e24e7f7359fa	172.31.1.0/24	public-b

Database subnets

When you run an Amazon RDS database attached to your Elastic Beanstalk environment, choose subnets for your database instances. For high availability, make the database multi-AZ and choose a subnet for each availability zone. To ensure that your application can connect to your database, run both in the same subnets.

The aws:ec2:vpc namespace

You can use the configuration options in the `aws:ec2:vpc` namespace to configure your environment's network settings.

The following configuration file uses options in this namespace to set the environment's VPC and subnets for a public-private configuration. In order to set the VPC ID in a configuration file, the file must be included in the application source bundle during environment creation. See Setting configuration options during environment creation for other methods of configuring these settings during environment creation.

Example .ebextensions/vpc.config – Public-private

```
option_settings:
   aws:ec2:vpc:
      VPCId: vpc-087a68c03b9c50c84
      AssociatePublicIpAddress: 'false'
      ELBScheme: public
      ELBSubnets: subnet-0fe6b36bcb0ffc462,subnet-032fe3068297ac5b2
      Subnets: subnet-026c6117b178a9c45,subnet-0839e902f656e8bd1
```

This example shows a public-public configuration, where the load balancer and EC2 instances run in the same public subnets.

Example .ebextensions/vpc.config – Public-public

```
option_settings:
   aws:ec2:vpc:
      VPCId: vpc-087a68c03b9c50c84
      AssociatePublicIpAddress: 'true'
      ELBScheme: public
      ELBSubnets: subnet-0fe6b36bcb0ffc462,subnet-032fe3068297ac5b2
      Subnets: subnet-0fe6b36bcb0ffc462,subnet-032fe3068297ac5b2
```

Migrating Elastic Beanstalk environments from EC2-Classic to a VPC

This topic describes different options for how to migrate your Elastic Beanstalk environments from an EC2-Classic network platform to an Amazon Virtual Private Cloud (Amazon VPC) network.

If you created your AWS account before December 4, 2013, you might have environments that use the EC2-Classic network configuration in some AWS Regions. All AWS accounts created on or after December 4, 2013 are already VPC-only in every AWS Region. The only exemptions are if Amazon EC2-Classic was enabled as a result of a support request.

Note

You can view the network configuration settings for your environment in the **Network configuration** category on the Configuration overview page of the Elastic Beanstalk console.

Why you should migrate

Amazon EC2-Classic will reach its end of standard support on August 15, 2022. To avoid interruptions to your workloads, we recommend that you migrate from Amazon EC2-Classic to a VPC before August 15, 2022. We also request that you don't launch any AWS resources on Amazon EC2-Classic in the future and use Amazon VPC instead.

When you migrate your Elastic Beanstalk environments from Amazon EC2-Classic to Amazon VPC, you must create a new AWS account. You must also re-create your AWS EC2-Classic environments in your new AWS account. No additional configuration work for your environments is required to use the default VPC. If the default VPC doesn't meet your requirements, manually create a custom VPC and associate it with your environments.

Alternatively, if your existing AWS account has resources that you can't migrate to a new AWS account, add a VPC into your current account. Then, configure your environments to use the VPC.

For more information, see the EC2-Classic Networking is Retiring - Here's How to Prepare blog post.

Migrate an environment from EC2-Classic into a new AWS account (recommended)

If you don't already have an AWS account that was created on or after December 4, 2013, create a new account. You will migrate your environments into this new account.

1. Your new AWS account provides a default VPC to its environments. If you don't need to create a custom VPC, skip to step 2.

   You can create a custom VPC in one of the following ways:

   - Create a VPC quickly using the Amazon VPC console wizard with one of the available configuration options. For more information, see Amazon VPC console wizard configurations.
   - Create a custom VPC on the Amazon VPC console if you have more specific requirements for your VPC. We recommend you do this, for example, if your use case requires a specific number of subnets. For more information, see VPCs and subnets.
   - Create a VPC using the elastic-beanstalk-samples repository on the GitHub website if you prefer to use AWS CloudFormation templates with your Elastic Beanstalk environments. This

repository includes AWS CloudFormation templates. For more information, see Using Elastic Beanstalk with Amazon VPC.

> **Note**
>
> You can also create a custom VPC at the same time you recreate the environment in your new AWS account using the create new environment wizard. If you use the wizard and choose to create a custom VPC, the wizard redirects you to the Amazon VPC console.

2. In your new AWS account, create a new environment. We recommend that the environment includes the same configuration as your existing environment in the AWS account that you're migrating from. You can do this by using one of the following approaches.

> **Note**
>
> If your new environment must use the same CNAME after you migrate, terminate the original environment on the EC2-Classic platform. This releases the CNAME for use. However, doing so can result in downtime for that environment and can also risk that another customer might select your CNAME between you terminating your EC2-Classic environment and creating the new one. For more information, see Terminate an Elastic Beanstalk environment.
> For environments that have their own proprietary domain name, the CNAME doesn't have this issue. You can just update your Domain Name System (DNS) to forward requests to your new CNAME.

- Use the create new environment wizard on the Elastic Beanstalk console. The wizard provides an option to create a custom VPC. If you don't choose to create a custom VPC, a default VPC is assigned.
- Use the Elastic Beanstalk Command Line Interface (EB CLI) to re-create your environment in your new AWS account. One of the examples in the **eb create** command description demonstrates the creation of an environment in a custom VPC. If you don't provide a VPC ID, the environment uses the default VPC.

  By using this approach, you can use a saved configurations file across the two AWS accounts. As a result, you don't need to manually enter all the configuration information.

However, you must save the configuration settings for the EC2-Classic environment that you're migrating with the eb config save command. Copy the saved configuration file to a new directory for the new account environment.

> **Note**
>
> You must edit some of the data in the saved configuration file before you can use it in the new account. You must also update information that pertains to your previous account with the correct data for your new account. For example, you must replace the Amazon Resource Name (ARN) of the AWS Identity and Access Management (IAM) role with the IAM role ARN for the new account.

If you use the eb create command with the `cfg`, the new environment is created using the specified saved configuration file. For more information, see Using Elastic Beanstalk saved configurations.

Migrate an environment from EC2-Classic within your same AWS account

Your existing AWS account might have resources that you can't migrate to a new AWS account. In this case you must re-create your environments and manually configure a VPC for every environment you create.

Migrate your environments to a custom VPC

Prerequisites

Before you begin, you must have a VPC. You can create a non-default (custom) VPC in one of the following ways:

- Create a VPC quickly using the Amazon VPC console wizard with one of the available configuration options. For more information, see Amazon VPC console wizard configurations.
- Create a custom VPC on the Amazon VPC console if you have more specific requirements for your VPC. We recommend you do this, for example, if your use case requires a specific number of subnets. For more information, see VPCs and subnets.
- Create a VPC using the elastic-beanstalk-samples repository on the GitHub website if you prefer to use AWS CloudFormation templates with your Elastic Beanstalk environments. This repository

includes AWS CloudFormation templates. For more information, see Using Elastic Beanstalk with Amazon VPC.

In the following steps, you use the generated VPC ID and subnet IDs when you configure the VPC in the new environment.

1. Create a new environment that includes the same configuration as your existing environment. You can do this by using one of the following approaches.

   Note

   The Saved Configurations feature can help you re-create your environments in the new account. This feature can save an environment's configuration, so you can apply it when you create or update other environments. For more information, see Using Elastic Beanstalk saved configurations.

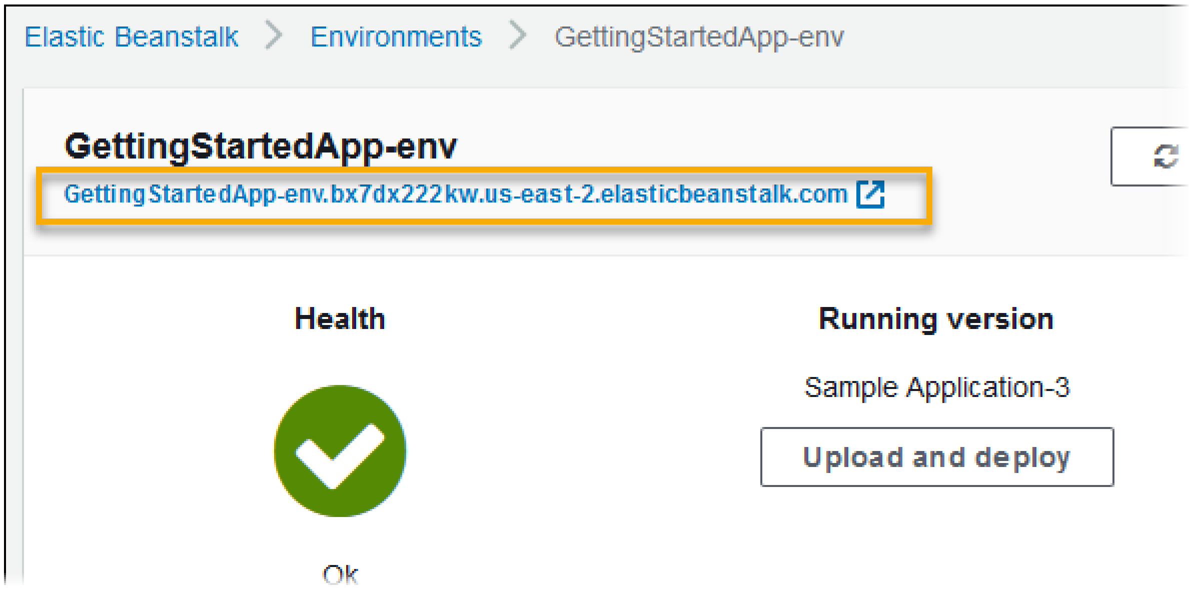
   - Using the Elastic Beanstalk console, apply a saved configuration from your EC2-Classic environment when you configure the new environment. This configuration will use the VPC. For more information, see Using Elastic Beanstalk saved configurations.
   - Using Elastic Beanstalk Command Line Interface (EB CLI), run the eb create command to re-create your environment. Provide the parameters of your original environment and the VPC identifier. One of the examples in the **eb create** command description shows how to create an environment in a custom VPC.
   - Use the AWS Command Line Interface (AWS CLI), and re-create your environment using the **elasticbeanstalk create-environment** command. Provide the parameters of your original environment with the VPC identifier. For instructions, see Creating Elastic Beanstalk environments with the AWS CLI.
2. Swap the CNAMEs of the existing environment with the new environment. This way, the new environment that you created can be referenced with the familiar address. You can use the EB CLI or the AWS CLI.
   - Using the EB CLI, swap the environment CNAMEs by running the **eb swap** command. For more information, see Setting up the EB command line interface (EB CLI) to manage Elastic Beanstalk.

- Using the AWS CLI, swap the environment CNAMEs with the elasticbeanstalk swap-environment-cnames command. For more information, see the AWS CLI Command Reference.

Your Elastic Beanstalk environment's Domain name

By default, your environment is available to users at a subdomain of `elasticbeanstalk.com`. When you create an environment, you can choose a hostname for your application. The subdomain and domain are autopopulated to *`region`*`.elasticbeanstalk.com`.

To route users to your environment, Elastic Beanstalk registers a CNAME record that points to your environment's load balancer. You can see URL of your environment's application with the current value of the CNAME in the environment overview page of the Elastic Beanstalk console.


Elastic Beanstalk > Environments > GettingStartedApp-env
GettingStartedApp-env
GettingStartedApp-env.bx7dx222kw.us-east-2.elasticbeanstalk.com
Health
Ok
Running version
Sample Application-3
Upload and deploy

Choose the URL on the overview page, or choose **Go to environment** on the navigation pane, to navigate to your application's web page.

You can change the CNAME on your environment by swapping it with the CNAME of another environment. For instructions, see Blue/Green deployments with Elastic Beanstalk.

If you own a domain name, you can use Amazon Route 53 to resolve it to your environment. You can purchase a domain name with Amazon Route 53, or use one that you purchase from another provider.

To purchase a domain name with Route 53, see Registering a New Domain in the *Amazon Route 53 Developer Guide*.

To learn more about using a custom domain, see Routing Traffic to an AWS Elastic Beanstalk Environment in the *Amazon Route 53 Developer Guide*.

Important

If you terminate an environment, you must also delete any CNAME mappings that you created, as other customers can reuse an available hostname. Be sure to delete DNS records that point to your terminated environment to prevent a *dangling DNS entry*. A dangling DNS entry can expose internet traffic destined for your domain to security vulnerabilities. It can also present other risks.

For more information, see Protection from dangling delegation records in Route 53 in the *Amazon Route 53 Developer Guide*. You can also learn more about dangling DNS entries in Enhanced Domain Protections for Amazon CloudFront Requests in the *AWS Security Blog*.

Configuring Elastic Beanstalk environments (advanced)

When you create an AWS Elastic Beanstalk environment, Elastic Beanstalk provisions and configures all of the AWS resources required to run and support your application. In addition to configuring your environment's metadata and update behavior, you can customize these resources by providing values for configuration options. For example, you may want to add an Amazon SQS queue and an alarm on queue depth, or you might want to add an Amazon ElastiCache cluster.

Most of the configuration options have default values that are applied automatically by Elastic Beanstalk. You can override these defaults with configuration files, saved configurations, command line options, or by directly calling the Elastic Beanstalk API. The EB CLI and Elastic Beanstalk console also apply recommended values for some options.

You can easily customize your environment at the same time that you deploy your application version by including a configuration file with your source bundle. When customizing the software on your instance, it is more advantageous to use a configuration file than to create a custom AMI because you do not need to maintain a set of AMIs.

When deploying your applications, you may want to customize and configure the software that your application depends on. These files could be either dependencies required by the application —for example, additional packages from the yum repository—or they could be configuration files such as a replacement for httpd.conf to override specific settings that are defaulted by AWS Elastic Beanstalk.

Topics

Configuration options

Elastic Beanstalk defines a large number of configuration options that you can use to configure your environment's behavior and the resources that it contains. Configuration options are organized into namespaces like `aws:autoscaling:asg`, which defines options for an environment's Auto Scaling group.

The Elastic Beanstalk console and EB CLI set configuration options when you create an environment, including options that you set explicitly, and recommended values defined by the client. You can also set configuration options in saved configurations and configuration files. If the same option is set in multiple locations, the value used is determined by the order of precedence.

Configuration option settings can be composed in text format and saved prior to environment creation, applied during environment creation using any supported client, and added, modified or removed after environment creation. For a detailed breakdown of all of the available methods for working with configuration options at each of these three stages, read the following topics:

- Setting configuration options before environment creation
- Setting configuration options during environment creation
- Setting configuration options after environment creation

For a complete list of namespaces and options, including default and supported values for each, see General options for all environments and Platform specific options.

Precedence

During environment creation, configuration options are applied from multiple sources with the following precedence, from highest to lowest:

- **Settings applied directly to the environment** – Settings specified during a create environment or update environment operation on the Elastic Beanstalk API by any client, including the Elastic Beanstalk console, EB CLI, AWS CLI, and SDKs. The Elastic Beanstalk console and EB CLI also apply recommended values for some options that apply at this level unless overridden.
- **Saved Configurations** – Settings for any options that are not applied directly to the environment are loaded from a saved configuration, if specified.

- **Configuration Files (.ebextensions) –** Settings for any options that are not applied directly to the environment, and also not specified in a saved configuration, are loaded from configuration files in the `.ebextensions` folder at the root of the application source bundle.

  Configuration files are executed in alphabetical order. For example, `.ebextensions/01run.config` is executed before `.ebextensions/02do.config`.
- **Default Values –** If a configuration option has a default value, it only applies when the option is not set at any of the above levels.

If the same configuration option is defined in more than one location, the setting with the highest precedence is applied. When a setting is applied from a saved configuration or settings applied directly to the environment, the setting is stored as part of the environment's configuration. These settings can be removed with the AWS CLI or with the EB CLI.

Settings in configuration files are not applied directly to the environment and cannot be removed without modifying the configuration files and deploying a new application version. If a setting applied with one of the other methods is removed, the same setting will be loaded from configuration files in the source bundle.

For example, say you set the minimum number of instances in your environment to 5 during environment creation, using either the Elastic Beanstalk console, a command line option, or a saved configuration. The source bundle for your application also includes a configuration file that sets the minimum number of instances to 2.

When you create the environment, Elastic Beanstalk sets the `MinSize` option in the `aws:autoscaling:asg` namespace to 5. If you then remove the option from the environment configuration, the value in the configuration file is loaded, and the minimum number of instances is set to 2. If you then remove the configuration file from the source bundle and redeploy, Elastic Beanstalk uses the default setting of 1.

Recommended values

The Elastic Beanstalk Command Line Interface (EB CLI) and Elastic Beanstalk console provide recommended values for some configuration options. These values can be different from the default values and are set at the API level when your environment is created. Recommended values allow Elastic Beanstalk to improve the default environment configuration without making backwards incompatible changes to the API.

For example, both the EB CLI and Elastic Beanstalk console set the configuration option for EC2 instance type (`InstanceType` in the `aws:autoscaling:launchconfiguration` namespace). Each client provides a different way of overriding the default setting. In the console you can choose a different instance type from a drop down menu on the **Configuration Details** page of the **Create New Environment** wizard. With the EB CLI, you can use the `--instance_type` parameter for **eb create**.

Because the recommended values are set at the API level, they will override values for the same options that you set in configuration files or saved configurations. The following options are set:

Elastic Beanstalk console

- Namespace: `aws:autoscaling:launchconfiguration`

  Option Names: `IamInstanceProfile`, `EC2KeyName`, `InstanceType`
- Namespace: `aws:autoscaling:updatepolicy:rollingupdate`

  Option Names: `RollingUpdateType` and `RollingUpdateEnabled`
- Namespace: `aws:elasticbeanstalk:application`

  Option Name: `Application Healthcheck URL`
- Namespace: `aws:elasticbeanstalk:command`

  Option Name: `DeploymentPolicy`, `BatchSize` and `BatchSizeType`
- Namespace: `aws:elasticbeanstalk:environment`

  Option Name: `ServiceRole`
- Namespace: `aws:elasticbeanstalk:healthreporting:system`

  Option Name: `SystemType` and `HealthCheckSuccessThreshold`
- Namespace: `aws:elasticbeanstalk:sns:topics`

  Option Name: `Notification Endpoint`
- Namespace: `aws:elasticbeanstalk:sqsd`

  Option Name: `HttpConnections`
- Namespace: `aws:elb:loadbalancer`

  Option Name: `CrossZone`

- Namespace: `aws:elb:policies`

  Option Names: `ConnectionDrainingTimeout` and `ConnectionDrainingEnabled`

EB CLI

- Namespace: `aws:autoscaling:launchconfiguration`

  Option Names: `IamInstanceProfile`, `InstanceType`
- Namespace: `aws:autoscaling:updatepolicy:rollingupdate`

  Option Names: `RollingUpdateType` and `RollingUpdateEnabled`
- Namespace: `aws:elasticbeanstalk:command`

  Option Name: `BatchSize` and `BatchSizeType`
- Namespace: `aws:elasticbeanstalk:environment`

  Option Name: `ServiceRole`
- Namespace: `aws:elasticbeanstalk:healthreporting:system`

  Option Name: `SystemType`
- Namespace: `aws:elb:loadbalancer`

  Option Name: `CrossZone`
- Namespace: `aws:elb:policies`

  Option Names: `ConnectionDrainingEnabled`

Setting configuration options before environment creation

AWS Elastic Beanstalk supports a large number of configuration options that let you modify the settings that are applied to resources in your environment. Several of these options have default values that can be overridden to customize your environment. Other options can be configured to enable additional features.

Elastic Beanstalk supports two methods of saving configuration option settings. Configuration files in YAML or JSON format can be included in your application's source code in a directory named

.ebextensions and deployed as part of your application source bundle. You create and manage configuration files locally.

Saved configurations are templates that you create from a running environment or JSON options file and store in Elastic Beanstalk. Existing saved configurations can also be extended to create a new configuration.

Note

Settings defined in configuration files and saved configurations have lower precedence than settings configured during or after environment creation, including recommended values applied by the Elastic Beanstalk console and EB CLI. See Precedence for details.

Options can also be specified in a JSON document and provided directly to Elastic Beanstalk when you create or update an environment with the EB CLI or AWS CLI. Options provided directly to Elastic Beanstalk in this manner override all other methods.

For a full list of available options, see Configuration options.

Methods

- Configuration files (.ebextensions)
- Saved configurations
- JSON document
- EB CLI configuration

Configuration files (.ebextensions)

Use .ebextensions to configure options that are required to make your application work, and provide default values for other options that can be overridden at a higher level of precedence. Options specified in .ebextensions have the lowest level of precedence and are overridden by settings at any other level.

To use configuration files, create a folder named .ebextensions at the top level of your project's source code. Add a file with the extension .config and specify options in the following manner:

```
option_settings:
  - namespace:  namespace
```

```
    option_name:  option name
    value:  option value
  - namespace:  namespace
    option_name:  option name
    value:  option value
```

For example, the following configuration file sets the application's health check url to `/health`:

`healthcheckurl.config`

```
option_settings:
  - namespace:  aws:elasticbeanstalk:application
    option_name:  Application Healthcheck URL
    value:  /health
```

In JSON:

```
{
 "option_settings" :
    [
      {
        "namespace" : "aws:elasticbeanstalk:application",
        "option_name" : "Application Healthcheck URL",
        "value" : "/health"
      }
    ]
}
```

This configures the Elastic Load Balancing load balancer in your Elastic Beanstalk environment to make an HTTP request to the path `/health` to each EC2 instance to determine if it is healthy or not.

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

Include the `.ebextensions` directory in your Application Source Bundle and deploy it to a new or existing Elastic Beanstalk environment.

Configuration files support several sections in addition to `option_settings` for customizing the software and files that run on the servers in your environment. For more information, see .Ebextensions.

Saved configurations

Create a saved configuration to save settings that you have applied to an existing environment during or after environment creation by using the Elastic Beanstalk console, EB CLI, or AWS CLI. Saved configurations belong to an application and can be applied to new or existing environments for that application.

Clients

- Elastic Beanstalk console
- EB CLI
- AWS CLI

Elastic Beanstalk console

To create a saved configuration (Elastic Beanstalk console)

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Save configuration**.
4. Use the on-screen dialog box to complete the action.

Saved configurations are stored in the Elastic Beanstalk S3 bucket in a folder named after your application. For example, configurations for an application named `my-app` in the us-west-2 region for account number 123456789012 can be found at `s3://elasticbeanstalk-us-west-2-123456789012/resources/templates/my-app`.

EB CLI

The EB CLI also provides subcommands for interacting with saved configurations under **eb config**:

To create a saved configuration (EB CLI)

1. Save the attached environment's current configuration:

```
~/project$ eb config save --cfg my-app-v1
```

The EB CLI saves the configuration to `~/project/.elasticbeanstalk/saved_configs/my-app-v1.cfg.yml`

2. Modify the saved configuration locally if needed.
3. Upload the saved configuration to S3:

```
~/project$ eb config put my-app-v1
```

AWS CLI

Create a saved configuration from a running environment with `aws elasticbeanstalk create-configuration-template`

To create a saved configuration (AWS CLI)

1. Identify your Elastic Beanstalk environment's environment ID with `describe-environments`:

```
$ aws elasticbeanstalk describe-environments --environment-name my-env
{
    "Environments": [
        {
            "ApplicationName": "my-env",
            "EnvironmentName": "my-env",
            "VersionLabel": "89df",
            "Status": "Ready",
            "Description": "Environment created from the EB CLI using \"eb create
\"",
            "EnvironmentId": "e-vcghmm2zwk",
            "EndpointURL": "awseb-e-v-AWSEBLoa-1JUM8159RA11M-43V6ZI1194.us-
west-2.elb.amazonaws.com",
            "SolutionStackName": "64bit Amazon Linux 2015.03 v2.0.2 running Multi-
container Docker 1.7.1 (Generic)",
            "CNAME": "my-env-nfptuqaper.elasticbeanstalk.com",
            "Health": "Green",
            "AbortableOperationInProgress": false,
            "Tier": {
                "Version": " ",
                "Type": "Standard",
                "Name": "WebServer"
            },
            "HealthStatus": "Ok",
            "DateUpdated": "2015-10-01T00:24:04.045Z",
            "DateCreated": "2015-09-30T23:27:55.768Z"
        }
    ]
}
```

2. Save the environment's current configuration with `create-configuration-template`:

   ```
   $ aws elasticbeanstalk create-configuration-template --environment-id e-vcghmm2zwk
    --application-name my-app --template-name v1
   ```

Elastic Beanstalk saves the configuration to your Elastic Beanstalk bucket in Amazon S3.

JSON document

If you use the AWS CLI to create and update environments, you can also provide configuration options in JSON format. A library of configuration files in JSON is useful if you use the AWS CLI to create and manage environments.

For example, the following JSON document sets the application's health check url to `/health`:

~/ebconfigs/healthcheckurl.json

```
[
  {
    "Namespace": "aws:elasticbeanstalk:application",
    "OptionName": "Application Healthcheck URL",
    "Value": "/health"
  }
]
```

EB CLI configuration

In addition to supporting saved configurations and direct environment configuration with **eb config** commands, the EB CLI has a configuration file with an option named `default_ec2_keyname` that you can use to specify an Amazon EC2 key pair for SSH access to the instances in your environment. The EB CLI uses this option to set the `EC2KeyName` configuration option in the `aws:autoscaling:launchconfiguration` namespace.

~/workspace/my-app/.elasticbeanstalk/config.yml

```
branch-defaults:
  master:
    environment: my-env
  develop:
    environment: my-env-dev
deploy:
  artifact: ROOT.war
global:
  application_name: my-app
  default_ec2_keyname: my-keypair
  default_platform: Tomcat 8 Java 8
  default_region: us-west-2
  profile: null
  sc: git
```

Setting configuration options during environment creation

When you create an AWS Elastic Beanstalk environment by using the Elastic Beanstalk console, EB CLI, AWS CLI, an SDK, or the Elastic Beanstalk API, you can provide values for configuration options to customize your environment and the AWS resources that are launched within it.

For anything other than a one-off configuration change, you can store configuration files locally, in your source bundle, or in Amazon S3.

This topic includes procedures for all of the methods to set configuration options during environment creation.

Clients

- In the Elastic Beanstalk console
- Using the EB CLI
- Using the AWS CLI

In the Elastic Beanstalk console

When you create an Elastic Beanstalk environment in the Elastic Beanstalk console, you can provide configuration options using configuration files, saved configurations, and forms in the **Create New Environment** wizard.

Methods

- Using configuration files (.ebextensions)
- Using a saved configuration
- Using the new environment wizard

Using configuration files (`.ebextensions`)

Include `.config` files in your application source bundle in a folder named `.ebextensions`.

For details about configuration files, see .Ebextensions.

```
~/workspace/my-app-v1.zip
|-- .ebextensions
|   |-- environmentvariables.config
|   `-- healthcheckurl.config
|-- index.php
`-- styles.css
```

Upload the source bundle to Elastic Beanstalk normally, during environment creation.

The Elastic Beanstalk console applies recommended values for some configuration options and has form fields for others. Options configured by the Elastic Beanstalk console are applied directly to the environment and override settings in configuration files.

Using a saved configuration

When you create a new environment using the Elastic Beanstalk console, one of the first steps is to choose a configuration. The configuration can be a **predefined configuration**, typically the latest version of a platform such as **PHP** or **Tomcat**, or it can be a **saved configuration**.

To apply a saved configuration during environment creation (Elastic Beanstalk console)

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.
3. In the navigation pane, find your application's name and choose **Saved configurations**.
4. Select the saved configuration you want to apply, and then choose **Launch environment**.

5. Proceed through the wizard to create your environment.

Saved configurations are application-specific. See Saved configurations for details on creating saved configurations.

Using the new environment wizard

Most of the standard configuration options are presented on the **Configure more options** page of the Create New Environment wizard. If you create an Amazon RDS database or configure a VPC for your environment, additional configuration options are available for those resources.

To set configuration options during environment creation (Elastic Beanstalk console)

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**.
3. Choose or create an application.
4. Choose **Actions**, and then choose **Create environment**.
5. Proceed through the wizard, and choose **Configure more options**.
6. Choose any of the **configuration presets**, and then choose **Edit** in one or more of the configuration categories to change a group of related configuration options.
7. When you are done making option selections, choose **Create environment**.

Any options that you set in the new environment wizard are set directly on the environment and override any option settings in saved configurations or configuration files (`.ebextensions`) that you apply. You can remove settings after the environment is created using the EB CLI or AWS CLI to allow the settings in saved configurations or configuration files to surface.

For details about the new environment wizard, see The create new environment wizard.

Using the EB CLI

Methods

- Using configuration files (.ebextensions)
- Using saved configurations
- Using command line options

Using configuration files (`.ebextensions`)

Include `.config` files in your project folder under `.ebextensions` to deploy them with your application code.

For details about configuration files, see .Ebextensions.

```
~/workspace/my-app/
|-- .ebextensions
|   |-- environmentvariables.config
|   `-- healthcheckurl.config
|-- .elasticbeanstalk
|   `-- config.yml
|-- index.php
`-- styles.css
```

Create your environment and deploy your source code to it with **eb create**.

```
~/workspace/my-app$ eb create my-env
```

Using saved configurations

To apply a saved configuration when you create an environment with **eb create**, use the `--cfg` option.

```
~/workspace/my-app$ eb create --cfg savedconfig
```

You can store the saved configuration in your project folder or in your Elastic Beanstalk storage location on Amazon S3. In the previous example, the EB CLI first looks for a saved configuration file named `savedconfig.cfg.yml` in the folder `.elasticbeanstalk/saved_configs/`. Do not include the file name extensions (`.cfg.yml`) when applying a saved configuration with `--cfg`.

```
~/workspace/my-app/
|-- .ebextensions
|   `-- healthcheckurl.config
|-- .elasticbeanstalk
|   |-- saved_configs
|   |   `-- savedconfig.cfg.yml
|   `-- config.yml
|-- index.php
```

```
`-- styles.css
```

If the EB CLI does not find the configuration locally, it looks in the Elastic Beanstalk storage location in Amazon S3. For details on creating, editing, and uploading saved configurations, see Saved configurations.

Using command line options

The EB CLI **eb create** command has several options that you can use to set configuration options during environment creation. You can use these options to add an RDS database to your environment, configure a VPC, or override recommended values.

For example, the EB CLI uses the `t2.micro` instance type by default. To choose a different instance type, use the `--instance_type` option.

```
$ eb create my-env --instance_type t2.medium
```

To create an Amazon RDS database instance and attach it to your environment, use the `--database` options.

```
$ eb create --database.engine postgres --database.username dbuser
```

If you leave out the environment name, database password, or any other parameters that are required to create your environment, the EB CLI prompts you to enter them.

See eb create for a full list of available options and usage examples.

Using the AWS CLI

When you use the `create-environment` command to create an Elastic Beanstalk environment with the AWS CLI, the AWS CLI does not apply any recommended values. All configuration options are defined in configuration files in the source bundle that you specify.

Methods

- Using configuration files (.ebextensions)
- Using a saved configuration
- Using command line options

Using configuration files (.ebextensions)

To apply configuration files to an environment that you create with the AWS CLI, include them in the application source bundle that you upload to Amazon S3.

For details about configuration files, see .Ebextensions.

```
~/workspace/my-app-v1.zip
|-- .ebextensions
|   |-- environmentvariables.config
|   `-- healthcheckurl.config
|-- index.php
`-- styles.css
```

To upload an application source bundle and create an environment with the AWS CLI

1. If you don't already have an Elastic Beanstalk bucket in Amazon S3, create one with `create-storage-location`.

   ```
   $ aws elasticbeanstalk create-storage-location
   {
       "S3Bucket": "elasticbeanstalk-us-west-2-123456789012"
   }
   ```

2. Upload your application source bundle to Amazon S3.

   ```
   $ aws s3 cp sourcebundle.zip s3://elasticbeanstalk-us-west-2-123456789012/my-app/
   sourcebundle.zip
   ```

3. Create the application version.

   ```
   $ aws elasticbeanstalk create-application-version --application-name my-app --
   version-label v1 --description MyAppv1 --source-bundle S3Bucket="elasticbeanstalk-
   us-west-2-123456789012",S3Key="my-app/sourcebundle.zip" --auto-create-application
   ```

4. Create the environment.

   ```
   $ aws elasticbeanstalk create-environment --application-name my-app --environment-
   name my-env --version-label v1 --solution-stack-name "64bit Amazon Linux 2015.03
    v2.0.0 running Tomcat 8 Java 8"
   ```

Using a saved configuration

To apply a saved configuration to an environment during creation, use the `--template-name` parameter.

```
$ aws elasticbeanstalk create-environment --application-name my-app --environment-name
 my-env --template-name savedconfig --version-label v1
```

When you specify a saved configuration, do not also specify a solution stack name. Saved configurations already specify a solution stack and Elastic Beanstalk will return an error if you try to use both options.

Using command line options

Use the `--option-settings` parameter to specify configuration options in JSON format.

```
$ aws elasticbeanstalk create-environment --application-name my-app --environment-name
 my-env --version-label v1 --template-name savedconfig --option-settings '[
  {
    "Namespace": "aws:elasticbeanstalk:application",
    "OptionName": "Application Healthcheck URL",
    "Value": "/health"
  }
]
```

To load the JSON from a file, use the `file://` prefix.

```
$ aws elasticbeanstalk create-environment --application-name my-app --environment-
name my-env --version-label v1 --template-name savedconfig --option-settings file://
healthcheckurl.json
```

Elastic Beanstalk applies option settings that you specify with the `--option-settings` option directly to your environment. If the same options are specified in a saved configuration or configuration file, `--option-settings` overrides those values.

Setting configuration options after environment creation

You can modify the option settings on a running environment by applying saved configurations, uploading a new source bundle with configuration files (`.ebextensions`), or using a JSON document. The EB CLI and Elastic Beanstalk console also have client-specific functionality for setting and updating configuration options.

When you set or change a configuration option, you can trigger a full environment update, depending on the severity of the change. For example, changes to options in the `aws:autoscaling:launchconfiguration`, such as `InstanceType`, require that the Amazon EC2 instances in your environment are reprovisioned. This triggers a rolling update. Other configuration changes can be applied without any interruption or reprovisioning.

You can remove option settings from an environment with EB CLI or AWS CLI commands. Removing an option that has been set directly on an environment at an API level allows settings in configuration files, which are otherwise masked by settings applied directly to an environment, to surface and take effect.

Settings in saved configurations and configuration files can be overridden by setting the same option directly on the environment with one of the other configuration methods. However, these can only be removed completely by applying an updated saved configuration or configuration file. When an option is not set in a saved configuration, in a configuration file, or directly on an environment, the default value applies, if there is one. See Precedence for details.

Clients

- The Elastic Beanstalk console
- The EB CLI
- The AWS CLI

The Elastic Beanstalk console

You can update configuration option settings in the Elastic Beanstalk console by deploying an application source bundle that contains configuration files, applying a saved configuration, or modifying the environment directly with the **Configuration** page in the environment management console.

Methods

- Using configuration files (.ebextensions)
- Using a saved configuration
- Using the Elastic Beanstalk console

Using configuration files (`.ebextensions`)

Update configuration files in your source directory, create a new source bundle, and deploy the new version to your Elastic Beanstalk environment to apply the changes.

For details about configuration files, see .Ebextensions.

To deploy a source bundle

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Changes made to configuration files will not override option settings in saved configurations or settings applied directly to the environment at the API level. See Precedence for details.

Using a saved configuration

Apply a saved configuration to a running environment to apply option settings that it defines.

To apply a saved configuration to a running environment (Elastic Beanstalk console)

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.
3. In the navigation pane, find your application's name and choose **Saved configurations**.
4. Select the saved configuration you want to apply, and then choose **Load**.
5. Select an environment, and then choose **Load**.

Settings defined in a saved configuration override settings in configuration files, and are overridden by settings configured using the environment management console.

See Saved configurations for details on creating saved configurations.

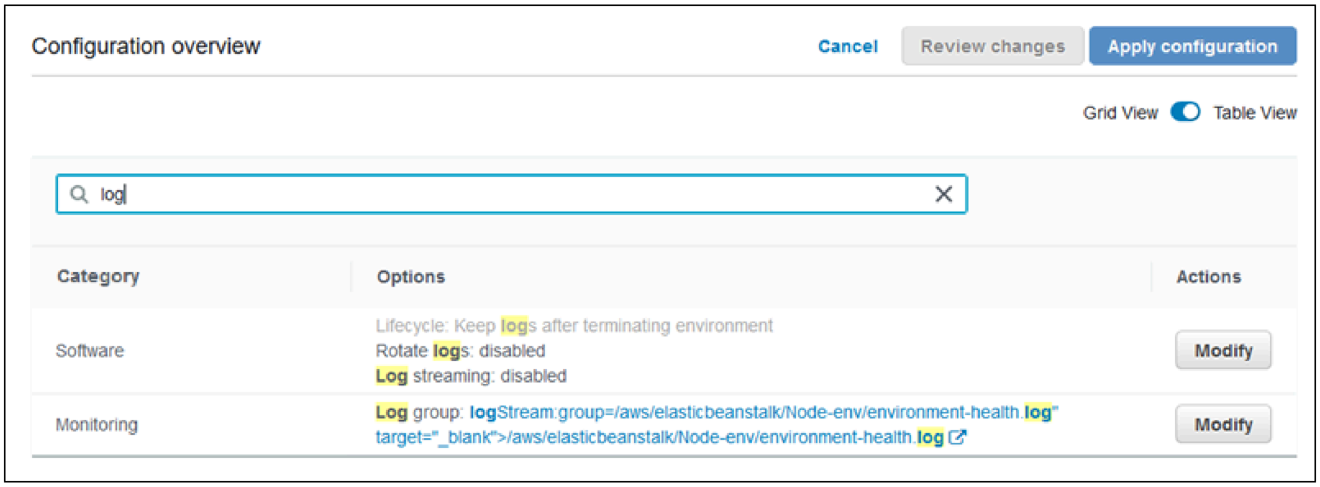
Using the Elastic Beanstalk console

The Elastic Beanstalk console presents many configuration options on the **Configuration** page for each environment.

To change configuration options on a running environment (Elastic Beanstalk console)

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. Find the configuration page you want to edit:
    - If you see the option you're interested in, or you know which configuration category it's in, choose **Edit** in the configuration category for it.
    - To look for an option, turn on **Table View**, and then enter search terms into the search box. As you type, the list gets shorter and shows only options that match your search terms.

      When you see the option you're looking for, choose **Edit** in the configuration category that contains it.



Category	Options	Actions
Software	Lifecycle: Keep logs after terminating environment Rotate logs: disabled Log streaming: disabled	Modify
Monitoring	Log group: logStream:group=/aws/elasticbeanstalk/Node-env/environment-health.log" target="_blank">/aws/elasticbeanstalk/Node-env/environment-health.log	Modify

5. Change settings, and then choose **Save**.
6. Repeat the previous two steps in additional configuration categories, as needed.
7. Choose **Apply**.

Changes made to configuration options in the environment management console are applied directly to the environment. These changes override settings for the same options in configuration files or saved configurations. For details, see Precedence.

For details about changing configuration options on a running environment using the Elastic Beanstalk console, see the topics under Configuring Elastic Beanstalk environments.

The EB CLI

You can update configuration option settings with the EB CLI by deploying source code that contains configuration files, applying settings from a saved configuration, or modifying the environment configuration directly with the **eb config** command.

Methods

- Using configuration files (.ebextensions)
- Using a saved configuration
- Using eb config
- Using eb setenv

Using configuration files (`.ebextensions`)

Include `.config` files in your project folder under `.ebextensions` to deploy them with your application code.

For details about configuration files, see .Ebextensions.

```
~/workspace/my-app/
|-- .ebextensions
|   |-- environmentvariables.config
|   `-- healthcheckurl.config
|-- .elasticbeanstalk
|   `-- config.yml
|-- index.php
`-- styles.css
```

Deploy your source code with **eb deploy**.

```
~/workspace/my-app$ eb deploy
```

Using a saved configuration

You can use the **eb config** command to apply a saved configuration to a running environment. Use the `--cfg` option with the name of the saved configuration to apply its settings to your environment.

```
$ eb config --cfg v1
```

In this example, `v1` is the name of a previously created and saved configuration file.

Settings applied to an environment with this command override settings that were applied during environment creation, and settings defined in configuration files in your application source bundle.

Using eb config

The EB CLI's **eb config** command lets you set and remove option settings directly on an environment by using a text editor.

When you run **eb config**, the EB CLI shows settings applied to your environment from all sources, including configuration files, saved configurations, recommended values, options set directly on the environment, and API defaults.

Note

eb config does not show environment properties. To set environment properties that you can read from within your application, use **eb setenv**.

The following example shows settings applied in the `aws:autoscaling:launchconfiguration` namespace. These settings include:

- Two recommended values, for `IamInstanceProfile` and `InstanceType`, applied by the EB CLI during environment creation.
- The option `EC2KeyName`, set directly on the environment during creation based on repository configuration.
- API default values for the other options.

```
ApplicationName: tomcat
```

```
DateUpdated: 2015-09-30 22:51:07+00:00
EnvironmentName: tomcat
SolutionStackName: 64bit Amazon Linux 2015.03 v2.0.1 running Tomcat 8 Java 8
settings:
...
aws:autoscaling:launchconfiguration:
    BlockDeviceMappings: null
    EC2KeyName: my-key
    IamInstanceProfile: aws-elasticbeanstalk-ec2-role
    ImageId: ami-1f316660
    InstanceType: t2.micro
...
```

To set or change configuration options with eb config

1. Run **eb config** to view your environment's configuration.

   ```
   ~/workspace/my-app/$ eb config
   ```

2. Change any of the setting values using the default text editor.

   ```
   aws:autoscaling:launchconfiguration:
       BlockDeviceMappings: null
       EC2KeyName: my-key
       IamInstanceProfile: aws-elasticbeanstalk-ec2-role
       ImageId: ami-1f316660
       InstanceType: t2.medium
   ```

3. Save the temporary configuration file and exit.
4. The EB CLI updates your environment configuration.

Setting configuration options with **eb config** overrides settings from all other sources.

You can also remove options from your environment with **eb config**.

To remove configuration options (EB CLI)

1. Run **eb config** to view your environment's configuration.

   ```
   ~/workspace/my-app/$ eb config
   ```

2. Replace any value shown with the string `null`. You can also delete the entire line containing the option that you want to remove.

   ```
   aws:autoscaling:launchconfiguration:
       BlockDeviceMappings: null
       EC2KeyName: my-key
       IamInstanceProfile: aws-elasticbeanstalk-ec2-role
       ImageId: ami-1f316660
       InstanceType: null
   ```

3. Save the temporary configuration file and exit.

4. The EB CLI updates your environment configuration.

Removing options from your environment with **eb config** allows settings for the same options to surface from configuration files in your application source bundle. See Precedence for details.

Using eb setenv

To set environment properties with the EB CLI, use **eb setenv**.

```
~/workspace/my-app/$ eb setenv ENVVAR=TEST
INFO: Environment update is starting.
INFO: Updating environment my-env's configuration settings.
INFO: Environment health has transitioned from Ok to Info. Command is executing on all
 instances.
INFO: Successfully deployed new configuration to environment.
```

This command sets environment properties in the `aws:elasticbeanstalk:application:environment` namespace. Environment properties set with **eb setenv** are available to your application after a short update process.

View environment properties set on your environment with **eb printenv**.

```
~/workspace/my-app/$ eb printenv
 Environment Variables:
     ENVVAR = TEST
```

The AWS CLI

You can update configuration option settings with the AWS CLI by deploying a source bundle that contains configuration files, applying a remotely stored saved configuration, or modifying the environment directly with the `aws elasticbeanstalk update-environment` command.

Methods

- Using configuration files (.ebextensions)
- Using a saved configuration
- Using command line options

Using configuration files (`.ebextensions`)

To apply configuration files to a running environment with the AWS CLI, include them in the application source bundle that you upload to Amazon S3.

For details about configuration files, see .Ebextensions.

```
~/workspace/my-app-v1.zip
|-- .ebextensions
|   |-- environmentvariables.config
|   `-- healthcheckurl.config
|-- index.php
`-- styles.css
```

To upload an application source bundle and apply it to a running environment (AWS CLI)

1. If you don't already have an Elastic Beanstalk bucket in Amazon S3, create one with `create-storage-location`:

   ```
   $ aws elasticbeanstalk create-storage-location
   {
       "S3Bucket": "elasticbeanstalk-us-west-2-123456789012"
   }
   ```

2. Upload your application source bundle to Amazon S3.

   ```
   $ aws s3 cp sourcebundlev2.zip s3://elasticbeanstalk-us-west-2-123456789012/my-app/
   sourcebundlev2.zip
   ```

3. Create the application version.

```
$ aws elasticbeanstalk create-application-version --application-
name my-app --version-label v2 --description MyAppv2 --source-bundle
 S3Bucket="elasticbeanstalk-us-west-2-123456789012",S3Key="my-app/
sourcebundlev2.zip"
```

4. Update the environment.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --version-label
 v2
```

Using a saved configuration

You can apply a saved configuration to a running environment with the `--template-name` option on the `aws elasticbeanstalk update-environment` command.

The saved configuration must be in your Elastic Beanstalk bucket in a path named after your application under `resources/templates`. For example, the `v1` template for the `my-app` application in the US West (Oregon) Region (us-west-2) for account 123456789012 is located at `s3://elasticbeanstalk-us-west-2-123456789012/resources/templates/my-app/v1`

To apply a saved configuration to a running environment (AWS CLI)

- Specify the saved configuration in an `update-environment` call with the `--template-name` option.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --template-
name v1
```

Elastic Beanstalk places saved configurations in this location when you create them with `aws elasticbeanstalk create-configuration-template`. You can also modify saved configurations locally and place them in this location yourself.

Using command line options

To change configuration options with a JSON document (AWS CLI)

1. Define your option settings in JSON format in a local file.

2. Run `update-environment` with the `--option-settings` option.

   ```
   $ aws elasticbeanstalk update-environment --environment-name my-env --option-
   settings file://~/ebconfigs/as-zero.json
   ```

In this example, `as-zero.json` defines options that configure the environment with a minimum and maximum of zero instances. This stops the instances in the environment without terminating the environment.

`~/ebconfigs/as-zero.json`

```
[
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "MinSize",
        "Value": "0"
    },
    {
        "Namespace": "aws:autoscaling:asg",
        "OptionName": "MaxSize",
        "Value": "0"
    },
    {
        "Namespace": "aws:autoscaling:updatepolicy:rollingupdate",
        "OptionName": "RollingUpdateEnabled",
        "Value": "false"
    }
]
```

Note

Setting configuration options with `update-environment` overrides settings from all other sources.

You can also remove options from your environment with `update-environment`.

To remove configuration options (AWS CLI)

- Run the `update-environment` command with the `--options-to-remove` option.

```
$ aws elasticbeanstalk update-environment --environment-name my-env --options-to-
remove Namespace=aws:autoscaling:launchconfiguration,OptionName=InstanceType
```

Removing options from your environment with `update-environment` allows settings for the same options to surface from configuration files in your application source bundle. If an option isn't configured using any of these methods, the API default value applies, if one exists. See Precedence for details.

General options for all environments

Namespaces

- aws:autoscaling:asg
- aws:autoscaling:launchconfiguration
- aws:autoscaling:scheduledaction
- aws:autoscaling:trigger
- aws:autoscaling:updatepolicy:rollingupdate
- aws:ec2:instances
- aws:ec2:vpc
- aws:elasticbeanstalk:application
- aws:elasticbeanstalk:application:environment
- aws:elasticbeanstalk:application:environmentsecrets
- aws:elasticbeanstalk:cloudwatch:logs
- aws:elasticbeanstalk:cloudwatch:logs:health
- aws:elasticbeanstalk:command
- aws:elasticbeanstalk:environment
- aws:elasticbeanstalk:environment:process:default
- aws:elasticbeanstalk:environment:process:process_name
- aws:elasticbeanstalk:environment:proxy:staticfiles
- aws:elasticbeanstalk:healthreporting:system
- aws:elasticbeanstalk:hostmanager

- aws:elasticbeanstalk:managedactions
- aws:elasticbeanstalk:managedactions:platformupdate
- aws:elasticbeanstalk:monitoring
- aws:elasticbeanstalk:sns:topics
- aws:elasticbeanstalk:sqsd
- aws:elasticbeanstalk:trafficsplitting
- aws:elasticbeanstalk:xray
- aws:elb:healthcheck
- aws:elb:loadbalancer
- aws:elb:listener
- aws:elb:listener:listener_port
- aws:elb:policies
- aws:elb:policies:policy_name
- aws:elbv2:listener:default
- aws:elbv2:listener:listener_port
- aws:elbv2:listenerrule:rule_name
- aws:elbv2:loadbalancer
- aws:rds:dbinstance

aws:autoscaling:asg

Configure your environment's Auto Scaling group. For more information, see the section called "Auto Scaling group".

Namespace: `aws:autoscaling:asg`

Name	Description	Default	Valid values
Availability Zones	Availability Zones (AZs) are distinct locations within an AWS Region that are engineered to be isolated from failures in other AZs. They provide inexpensive, low-latency network connectivity to other AZs in the same	Any	Any Any 1 Any 2 Any 3

Name	Description	Default	Valid values
	Region. Choose the number of AZs for your instances.		
Cooldown	Cooldown periods help prevent Amazon EC2 Auto Scaling from initiating additional scaling activities before the effects of previous activitie s are visible. A cooldown period is the amount of time, in seconds, after a scaling activity completes before another scaling activity can start.	`360`	`0` to `10000`

Custom Availability Zones	Define the AZs for your instances.	None	`us-east-1a`
			
			`us-east-1b`
			
			`us-east-1c`
			
			`us-east-1d`
			
			`us-east-1e`
			
			`eu-central-1`

EnableCap acityReba lancing	Specifies whether to enable the Capacity Rebalancing feature for Spot Instances in your Auto Scaling Group. For more information, see Capacity Rebalancing in the *Amazon EC2 Auto Scaling User Guide*.
	
	This option is only relevant when `EnableSpot` is set to `true` in the `aws:ec2:instances` namespace, and there is at least one Spot Instance in your Auto Scaling group.
		`false`	`true`
			
			`false`

Name	Description	Default	Valid values
MinSize	The minimum number of instances that you want in your Auto Scaling group.	1	1 to `10000`
MaxSize	The maximum number of instances that you want in your Auto Scaling group.	4	1 to `10000`

aws:autoscaling:launchconfiguration

Configure the Amazon Elastic Compute Cloud (Amazon EC2) instances for your environment.

The instances that are used for your environment are created using either an Amazon EC2 launch template or an Auto Scaling group launch configuration resource. The following options work with both of these resource types.

For more information, see the section called "Amazon EC2 instances". You can also reference more information about Amazon Elastic Block Store (EBS) in Amazon EBS chapter in the *Amazon EC2 User Guide*.

Namespace: `aws:autoscaling:launchconfiguration`

Name	Description	Default	Valid values

DisableDefaultEC2SecurityGroup	When set to the default value of `false`, Elastic Beanstalk creates a default security group that allows traffic from the internet or load balancer on the standard ports for HTTP (80). It attaches this security group to the EC2 instances of the environment when it creates the environment.
	
	When set to `true` Elastic Beanstalk will not assign the default security group to the EC2 instances for a
		`false`	`true`
			
			`false`

Name	Description	Default	Valid values
	new environment. For an existing environment, Elastic Beanstalk will unassign the default EC2 security group from your environment's EC2 instances. As a result, you must also set the following configurations: • The `SecurityGroups` option of this namespace will require at least one value to define your custom security group(s). • For environments with a load balancer, you will also need to set the `SecurityGroups` options in another namespace to configure custom security groups for the load balancer. For application load balancers, set the option in the aws:elbv2:loadbalancer namespace . For classic load balancers, set the option in the aws:elb:loadbalancer namespace. • For more information, see Managing EC2 security groups. If a value is specified for EC2KeyNam e in an environment that has `DisableDefaultEC2SecurityGroup` set to `true` a default security group will not be associated with the EC2 instances.		

Name	Description	Default	Valid values

DisableIMDSv1	Set to `true` to disable Instance Metadata Service Version 1 (IMDSv1) and enforce IMDSv2.
	
	Set to `false` to enable both IMDSv1 and IMDSv2.
	
	The instances for your environment default as follows, based on the platform operating system:
	
	• *Windows server, AL2 and earlier* – enable both IMDSv1 and IMDSv2 (DisableIMDSv1 defaults to `false`)
	• *AL2023* – enable only IMDSv2 (DisableIMDSv1 defaults to `true`)
	
	For more information, see Configuring the instance metadata service.
	
	Important
	This option setting can cause Elastic Beanstalk to migrate an existing environment with launch configurations to launch templates. Doing so requires the necessary permissions to manage launch templates. These permissions are included in our managed policy. If you use custom policies instead of our managed policies, environment creation or updates might
		`false` – platforms based on Windows server, Amazon Linux 2 and earlier
		
		`true` – platforms based on Amazon Linux 2023
			`true`
			
			`false`

Name	Description	Default	Valid values
	fail when you update your environment configura tion. For more information and other considerations, see Migrating your Elastic Beanstalk environment to launch templates .		
EC2KeyNam e	You can use a key pair to securely log into your EC2 instance. If a value is specified for `EC2KeyNam e` in an environment that has `DisableDefaultEC2SecurityGr oup` set to `true` a default security group will not be associated with the EC2 instances. **Note** If you use the Elastic Beanstalk console to create an environment, you can't set this option in a configuration file. The console overrides this option with a recommended value.	None	

Name	Description	Default	Valid values
IamInstan ceProfile	An instance profile enables AWS Identity and Access Managemen t (IAM) users and AWS services to access temporary security credentia ls to make AWS API calls. Specify the instance profile's name or its ARN. Examples: • `aws-elasticbeanstalk-ec2-role` • `arn:aws:iam::123456789012:instance-profile/aws-elasticbeanstalk-ec2-role` **Note** If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file. The console and EB CLI override this option with a recommended value.	None	Instance profile name or ARN.
ImageId	You can override the default Amazon Machine Image (AMI) by specifying your own custom AMI ID. Example: `ami-1f316660`	None	

Name	Description	Default	Valid values
InstanceType	The instance type that's used to run your application in an Elastic Beanstalk environment. **Important** The `InstanceType` option is obsolete. It's replaced by the newer and more powerful `InstanceTypes` option in the `aws:ec2:instances` namespace. You can use this new option to specify a list of one or more instance types for your environment. The first value on that list is equivalent to the value of the `InstanceType` option that's included in the `aws:autoscaling:launchconfiguration` namespace that's described here. We recommend that you specify instance types by using the new option. If specified, the new option takes precedence over the previous one. For more information, see the section called "The aws:ec2:instances namespace". The instance types that are available depend on the Availability Zones and Region used. If you choose a subnet, the Availability Zone that contains	Varies by account and Region.	One EC2 instance type. Varies by account, Region, and Availability Zone. You can obtain a list of Amazon EC2 instance types filtered by these values. For more information, see Available instance types in the *Amazon EC2 User Guide*.

Name	Description	Default	Valid values
	that subnet determines the available instance types. • Elastic Beanstalk doesn't support Amazon EC2 Mac instance types. • For more information about Amazon EC2 instance families and types, see Instance types in the *Amazon EC2 User Guide*. • For more information on the available instance types across Regions, see Available instance types in the *Amazon EC2 User Guide*. **Note** If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file. The console and EB CLI override this option with a recommended value.		

Name	Description	Default	Valid values

LaunchTemplateTagPropagationEnabled	Set to `true` to enable the propagation of environment tags to the launch templates for specific resources provisioned to the environment.
	
	Elastic Beanstalk can only propagate tags to launch templates for the following resources:
	
	• EBS volumes
	• EC2 instances
	• EC2 network interfaces
	• AWS CloudFormation launch templates that define a resource
	
	This constraint exists because CloudFormation only allows tags on template creation for specific resources. For more information see TagSpecification in the *AWS CloudFormation User Guide*.
	
	Important
	
	• Changing this option value from `false` to `true` for an existing environment may be a breaking change for previously existing tags.
	• When this feature is enabled, the propagation of tags will require EC2 replacement, which can result in downtime. You can
		`false`	`true`
			
			`false`

Name	Description	Default	Valid values
	enable *rolling updates* to apply configuration changes in batches and prevent downtime during the update process. For more informati on, see Configuration changes. For more information about launch templates, see the following: • Launch templates in the *Amazon EC2 Auto Scaling User Guide* • Working with templates in the *AWS CloudFormation User Guide* • Elastic Beanstalk template snippets in the *AWS CloudFormation User Guide* For more information about this option, see Tag propagation to launch templates.		

Monitorin gInterval	The interval (in minutes) that you want Amazon CloudWatch metrics to be returned at.	`5 minute`	`1 minute`
			
			`5 minute`

Name	Description	Default	Valid values
SecurityGroups	Lists the Amazon EC2 security group IDs to assign to the EC2 instances in the Auto Scaling group to define firewall rules for the instances. Use this option along with `DisableDefaultEC2SecurityGroup` to attach your own custom security groups that define firewall rules for the EC2 instances. For more information, see Load balanced (multi-instance) environments. **Important** You may need to complete some additional configuration to prevent incoming traffic to your EC2 instances from being blocked. This only applies to multi-instance environments with custom EC2 security groups. The EC2 security groups must include an inbound rule that grants access to traffic routed from the load balancer. For more information, see Managing EC2 security groups in multi-instance environments. You can provide a single string of comma-separated values that contain existing Amazon EC2 security groups	`elasticbeanstalk-default`	

Name	Description	Default	Valid values
	IDs or references to AWS::EC2::Security Group resources created in the template. You must provide at least one value for this option if `DisableDefaultEC2SecurityGroup` for this namespace is set to `true`.		

Name	Description	Default	Valid values
SSHSource Restriction	Used to lock down SSH access to an environment. For example, you can lock down SSH access to the EC2 instances so that only a bastion host can access the instances in the private subnet. This string takes the following form: *`protocol`*`,` *`fromPort`*`,` *`toPort`*`,` *`source_restriction`* *`protocol`* The protocol for the ingress rule. *`fromPort`* The starting port number. *`toPort`* The ending port number. *`source_restriction`* The Classless Inter-Domain Routing (CIDR) range or the security group that traffic must route through. Specify the security group with the security group ID. To specify a security group from another account, include the AWS account ID before the security group ID, separated by a forward slash. The other account must be in the same AWS Region. Note the syntax: *`aws-accou`*	None	

Name	Description	Default	Valid values
	`nt-id` /*`security-group-id`*. For example: **`123456789012 /sg-99999999`** **Examples:** • `tcp, 22, 22, 54.240.196.185/32` • `tcp, 22, 22, my-security-group-id` • `tcp, 22, 22, 123456789012/their-security-group-id`		

Name	Description	Default	Valid values
BlockDeviceMappings	Attach additional Amazon EBS volumes or instance store volumes on all of the instances in the Auto Scaling group. **Important** This option setting can cause Elastic Beanstalk to migrate an existing environment with launch configurations to launch templates. Doing so requires the necessary permissions to manage launch templates. These permissions are included in our managed policy. If you use custom policies instead of our managed policies, environment creation or updates might fail when you update your environment configuration. For more information and other considerations, see Migrating your Elastic Beanstalk environment to launch templates . When mapping instance store volumes, you only need to map the device name to a volume name. However, we recommend, when mapping Amazon EBS volumes, you additionally specify some or all of the	None	• size — must be between 500 and 16384 GiB • throughput — must be between 125 and 1000 mebibytes per second (MiB/s)

Name	Description	Default	Valid values
	following fields (each field must be separated by a colon): • snapshot ID • size, in GB • delete on terminate (`true` or `false`) • storage type (only for `gp3`, `gp2`, `standard`, `st1`, `sc1`, or `io1`) • IOPS (only for `gp3` or `io1`) • throughput (only for `gp3`) The following example attaches three Amazon EBS volumes, one blank 100GB gp2 volume and one snapshot, one blank 20GB io1 volume with 2000 provisioned IOPS, and an instance store volume `ephemeral0` . Multiple instance store volumes can be attached if the instance type supports it. `/dev/sdj=:100:true:gp2,/dev/sdh=snap-51eef269,/dev/sdi=:20:true:io1:2000,/dev/sdb=ephemeral0`		

Name	Description	Default	Valid values
RootVolumeType	Volume type (magnetic, general purpose SSD or provisioned IOPS SSD) to use for the root Amazon EBS volume attached to the EC2 instances for your environment. **Important** This option setting can cause Elastic Beanstalk to migrate an existing environment with launch configurations to launch templates. Doing so requires the necessary permissions to manage launch templates. These permissions are included in our managed policy. If you use custom policies instead of our managed policies, environment creation or updates might fail when you update your environment configuration. For more information and other considerations, see [Migrating your Elastic Beanstalk environment to launch templates]().	Varies by platform.	`standard` for magnetic storage. `gp2` or `gp3` for general purpose SSD. `io1` for provisioned IOPS SSD.

Name	Description	Default	Valid values
RootVolumeSize	The storage capacity of the root Amazon EBS volume in whole GB. Required if you set `RootVolumeType` to provisioned IOPS SSD. For example, `"64"`.	Varies per platform for magnetic storage and general purpose SSD. None for provisioned IOPS SSD.	`10` to `16384` GB for general purpose and provisioned IOPS SSD. `8` to `1024` GB for magnetic.
RootVolumeIOPS	The desired input/output operations per second (IOPS) for a provisioned IOPS SSD root volume or for a general purpose `gp3` SSD root volume. The maximum ratio of IOPS to volume size is 500 to 1. For example, a volume with 3000 IOPS must be at least 6 GiB.	None	`100` to `20000` for io1 provisioned IOPS SSD root volumes. `3000` to `16000` for general purpose `gp3` SSD root volumes.
RootVolumeThroughput	The desired throughput of mebibytes per second (MiB/s) to provision for the Amazon EBS root volume attached to your environment's EC2 instance. **Note** This option is only applicable to `gp3` storage types.	None	`125` to `1000`

aws:autoscaling:scheduledaction

Configure scheduled actions for your environment's Auto Scaling group. For each action, specify a `resource_name` in addition to the option name, namespace, and value for each setting. See The aws:autoscaling:scheduledaction namespace for examples.

Namespace: `aws:autoscaling:scheduledaction`

Name	Description	Default	Valid values
StartTime	For one-time actions, choose the date and time to run the action. For recurrent actions, choose when to activate the action.	None	A ISO-8601 timestamp unique across all scheduled scaling actions.
EndTime	A date and time in the future (in the UTC/GMT time zone) when you want the scheduled scaling action to stop repeating. If you don't specify an **EndTime**, the action recurs according to the `Recurrence` expression. Example: `2015-04-28T04:07:2Z` When a scheduled action ends, Amazon EC2 Auto Scaling doesn't automatically revert to its previous settings. Configure a second scheduled action to return to the original settings as needed.	None	A ISO-8601 timestamp unique across all scheduled scaling actions.
MaxSize	The maximum instance count to apply when the action runs.	None	`0` to `10000`
MinSize	The minimum instance count to apply when the action runs.	None	`0` to `10000`
DesiredCapacity	Set the initial desired capacity for the Auto Scaling group. After the scheduled action is applied, triggers adjust the desired capacity based on their settings.	None	`0` to `10000`

Name	Description	Default	Valid values
Recurrence	The frequency that you want the scheduled action to occur at. If you don't specify a recurrence, then the scaling action occurs only once, as specified by the `StartTime` .	None	A Cron expression.
Suspend	Set to `true` to deactivate a recurrent scheduled action temporarily.	`false`	`true` `false`

aws:autoscaling:trigger

Configure scaling triggers for your environment's Auto Scaling group.

Note

Three options in this namespace determine how long the metric for a trigger can remain beyond its defined limits before the trigger initates. These options are related as follows:

`BreachDuration = Period * EvaluationPeriods`

The default values for these options (5, 5, and 1, respectively) satisfy this equation. If you specify inconsistent values, Elastic Beanstalk might modify one of the values so that the equation is still satisfied.

Namespace: `aws:autoscaling:trigger`

Name	Description	Default	Valid values
BreachDuration	The amount of time, in minutes, a metric can be beyond its defined limit (as specified in the `UpperThreshold` and `LowerThreshold`) before the trigger is invoked.	5	`1` to `600`

Name	Description	Default	Valid values
LowerBreachScaleIncrement	How many Amazon EC2 instances to remove when performing a scaling activity.	`-1`	
LowerThreshold	If the measurement falls below this number for the breach duration, a trigger is invoked.	`2000000`	`0` to `20000000`

MeasureName	The metric that's used for your Auto Scaling trigger.
	
	Note
	`HealthyHostCount` , `UnhealthyHostCount` and `TargetResponseTime` are only applicable for environments with a dedicated load balancer. These aren't valid metric values for environments configured with a shared load balancer. For more information about load balancer types, see Load balancer for your Elastic Beanstalk environment.
		`NetworkOut`	`CPUUtilization`
			`NetworkIn`
			`NetworkOut`
			`DiskWriteOps`
			`DiskReadBytes`
			`DiskReadOps`
			`DiskWriteBytes`
			`Latency`
			`RequestCount`
			`HealthyHostCount`
			`UnhealthyHostCount`
			`TargetResponseTime`

Name	Description	Default	Valid values
Period	Specifies how frequently Amazon CloudWatch measures the metrics for your trigger. The value is the number of minutes between two consecutive periods.	5	1 to `600`
EvaluationPeriods	The number of consecutive evaluation periods that's used to determine if a breach is occurring.	1	1 to `600`

Statistic	The Statistic the trigger uses, such as `Average`.	`Average`	`Minimum`
			
			`Maximum`
			
			`Sum`
			
			`Average`

Unit	The unit for the trigger measurement, such as `Bytes`.	`Bytes`	`Seconds`
			
			`Percent`
			
			`Bytes`
			
			`Bits`
			
			`Count`
			
			`Bytes/Second`
			
			`Bits/Second`
			
			`Count/Second`
			
			`None`

UpperBreachScaleIncrement	Specifies how many Amazon EC2 instances to add when performing a scaling activity.	1	

Name	Description	Default	Valid values
UpperThreshold	If the measurement is higher than this number for the breach duration, a trigger is invoked.	`6000000`	`0` to `20000000`

aws:autoscaling:updatepolicy:rollingupdate

Configure rolling updates your environment's Auto Scaling group.

Namespace: `aws:autoscaling:updatepolicy:rollingupdate`

Name	Description	Default	Valid values
MaxBatchSize	The number of instances included in each batch of the rolling update.	One-third of the minimum size of the Auto Scaling group, rounded to the next highest integer.	`1` to `10000`
MinInstancesInService	The minimum number of instances that must be in service within the Auto Scaling group while other instances are terminated.	The minimum size of the Auto Scaling group or one fewer than the maximum size of the Auto Scaling group, whichever is lower.	`0` to 9999
RollingUpdateEnabled	If `true`, it enables rolling updates for an environment. Rolling updates are useful when you need to make small,	`false`	`true` `false`

Name	Description	Default	Valid values
	frequent updates to your Elastic Beanstalk software application and you want to avoid application downtime. Setting this value to true automatically enables the `MaxBatchSize` , `MinInstancesInService` , and `PauseTime` options. Setting any of those options also automatically sets the `RollingUpdateEnabled` option value to `true`. Setting this option to `false` disables rolling updates. **Note** If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in		

Name	Description	Default	Valid values
	a configuration file. The console and EB CLI override this option with a recommended value.		

Name	Description	Default	Valid values

RollingUpdateType	This includes three types: time-based rolling updates, health-based rolling updates, and immutable updates.
	
	Time-based rolling updates apply a PauseTime between batches. Health-based rolling updates wait for new instances to pass health checks before moving on to the next batch. Immutable updates launch a full set of instances in a new Auto Scaling group.
	
	Note
	If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file.
		`Time`	`Time`
			
			`Health`
			
			`Immutable`

Name	Description	Default	Valid values
	The console and EB CLI override this option with a recommended value.		
PauseTime	The amount of time (in seconds, minutes, or hours) the Elastic Beanstalk service waits after it completed updates to one batch of instances and before it continues on to the next batch.	Automatically computed based on instance type and container.	`PT0S`* (0 seconds) to `PT1H` (1 hour)
Timeout	The maximum amount of time (in minutes or hours) to wait for all instances in a batch of instances to pass health checks before canceling the update.	`PT30M` (30 minutes)	`PT5M`* (5 minutes) to `PT1H` (1 hour) *ISO8601 duration format: `PT#H#M#S` where each # is the number of hours, minutes, and/or seconds, respectively.

aws:ec2:instances

Configure your environment's instances, including Spot options. This namespace complements `aws:autoscaling:launchconfiguration` and `aws:autoscaling:asg`.

For more information, see the section called "Auto Scaling group".

Namespace: `aws:ec2:instances`

Name	Description	Defaul	Valid values

EnableSpot	Enable Spot Instance requests for your environment. When `false`, some options in this namespace don't take effect.
	
	Important
	This option setting can cause Elastic Beanstalk to migrate an existing environment with launch configurations to launch templates. Doing so requires the necessary permissions to manage launch templates. These permissions are included in our managed policy. If you use custom policies instead of our managed policies, environment creation or updates might fail when you update your environment configuration. For more information and other considerations, see Migrating your Elastic Beanstalk environment to launch templates .
		`false`	`true`
			
			`false`

InstanceTypes	A comma-separated list of instance types that you want your environment to use (for example, `t2.micro`, `t3.micro`).	A list of two	One to forty EC2 instance types. We recommend at least two.

Name	Description	Defaul	Valid values
	When `EnableSpot` is `true` and `SpotAllocationStrategy` is set to `capacity-optimized-prioritized`, the list of values specified in this option determines the instance type priority for the Spot Instance allocation strategy. When Spot Instances are not activated (`EnableSpot` is `false`), only the first instance type on the list is used. The first instance type on the list in this option is equivalent to the value of the `InstanceType` option in the `aws:autoscaling:launchconfiguration` namespace. We don't recommend using the latter option because it's obsolete. If you specify both, the first instance type on the list in the `InstanceTypes` option is used, and `InstanceType` is ignored. The instance types that are available depend on the Availability Zones and Region used. If you choose a subnet, the Availability Zone that contains that subnet determines the available instance types. • Elastic Beanstalk doesn't support Amazon EC2 Mac instance types. • For more information about Amazon EC2 instance families and	instan types. Varies by accour and Regior	Varies by account, Region, and Availability Zone. You can obtain a list of Amazon EC2 instance types filtered by these values. For more information, see Available instance types in the *Amazon EC2 User Guide*. The instance types must all be part of the same architecture (`arm64`, `x86_64`, `i386`). `SupportedArchitectures` is also part of this namespace. If you provide any values for `SupportedArchitectures`, the value(s) you enter for `InstanceTypes` must belong to one, and only one, of the architectures you provide for `SupportedArchitectures`.

Name	Description	Defaul	Valid values
	types, see Instance types in the *Amazon EC2 User Guide*. • For more information on the available instance types across Regions, see Available instance types in the *Amazon EC2 User Guide*. **Note** Some older AWS accounts might provide Elastic Beanstalk with default instance types that don't support Spot Instances (for example, t1.micro). If you activate Spot Instance requests and you get an error about an instance type that doesn’t support Spot, be sure to configure instance types that support Spot. To choose Spot Instance types, use the Spot Instance Advisor. When you update your environme nt configuration and remove one or more instance types from the `InstanceTypes` option, Elastic Beanstalk terminates any Amazon EC2 instances running on any of the removed instance types. Your environment's Auto Scaling group		

Name	Description	Defaul	Valid values
	then launches new instances, as necessary to complete the desired capacity, using your current specified instance types.		

SpotAllocationStrategy	Specifies the [spot instance allocation strategy](#) that determines how Spot Instances are allocated from the available spot capacity pools.
	
	If set to `capacity-optimized-prioritized` , the order of the values in `InstanceTypes` sets the instance type priority for allocation.
	
	This option is relevant only when `EnableSpot` is `true`.
		`capac optim`	`capacity-optimized`
			
			`price-capacity-optimized`
			
			`capacity-optimized-prioritized`
			
			`lowest-price`

SpotFleetOnDemandBase	The minimum number of On-Demand Instances that your Auto Scaling group provisions before considering Spot Instances as your environment scales up. This option is relevant only when `EnableSpot` is `true`.	`0`	`0` to `MaxSize` option in [`aws:autoscaling:asg`](#) namespace

Name	Description	Defaul	Valid values
SpotFleetOnDemandAboveBasePercentage	The percentage of On-Demand Instances as part of additional capacity that your Auto Scaling group provisions beyond the `SpotOnDemandBase` instances. This option is relevant only when `EnableSpot` is `true`.	`0` for a single-instance enviro nt `70` for a load-balanced enviro nt	`0` to `100`

SpotMaxPrice	The maximum price per unit hour, in USD, that you're willing to pay for a Spot Instance. For recommendations about maximum price options for Spot Instances, see [Spot Instance pricing history](#) in the *Amazon EC2 User Guide*.
	
	This option is relevant only when `EnableSpot` is `true`.
		On-Demar price, for each instan type. The option value in this case is `null`.	`0.001` to `20.0`
			
			`null`

Name	Description	Defaul	Valid values

SupportedArchitectures	A comma-separated list of EC2 instance architecture types that you'll use for your environment.
	
	Elastic Beanstalk supports instance types based on the following processor architectures:
	
	• AWS Graviton 64-bit Arm architecture (arm64)
	• 64-bit architecture (x86_64)
	• 32-bit architecture (i386)
	
	For more information about processor architecture and Amazon EC2 instance types see the section called "Amazon EC2 instance types".
		None	`arm64`
			
			`x86_64`
			
			`i386`
			
			Note
			The 32-bit architecture `i386` is not supported by the majority of Elastic Beanstalk platforms. We recommended that you choose the `x86_64` or `arm64` architecture types instead.

aws:ec2:vpc

Configure your environment to launch resources in a custom Amazon Virtual Private Cloud (Amazon VPC). If you don't configure settings in this namespace, Elastic Beanstalk launches resources in the default VPC.

Namespace: `aws:ec2:vpc`

Name	Description	Default	Valid values
VPCId	The ID for your Amazon VPC.	None	

Name	Description	Default	Valid values
Subnets	The IDs of the Auto Scaling group subnet or subnets. If you have multiple subnets, specify the value as a single comma-separated string of subnet IDs (for example, `"subnet-11111111,subnet-222 22222"`).	None	
ELBSubnets	The IDs of the subnet or subnets for the elastic load balancer. If you have multiple subnets, specify the value as a single comma-separated string of subnet IDs (for example, `"subnet-11111111,s ubnet-22222222"`).	None	
ELBScheme	Specify `internal` if you want to create an internal load balancer in your Amazon VPC so that your Elastic Beanstalk application can't be accessed from outside your Amazon VPC. If you specify a value other than `public` or `internal`, Elastic Beanstalk ignores the value.	`public`	`public` `internal`
DBSubnets	Contains the IDs of the database subnets. This is only used if you want to add an Amazon RDS DB Instance as part of your application. If you have multiple subnets, specify the value as a single comma-sep arated string of subnet IDs (for example, `"subnet-1 1111111,subnet-22222222"`).	None	

Name	Description	Default	Valid values

AssociatePublicIpAddress	Specifies whether to launch instances with public IP addresses in your Amazon VPC. Instances with public IP addresses don't require a NAT device to communicate with the Internet. You must set the value to `true` if you want to include your load balancer and instances in a single public subnet.
	
	This option has no effect on a single-instance environment, which always has a single Amazon EC2 instance with an Elastic IP address. The option is relevant to load-balanced, scalable environments.
		None	`true`
			
			`false`

aws:elasticbeanstalk:application

Configure a health check path for your application. For more information, see Basic health reporting.

Namespace: `aws:elasticbeanstalk:application`

Name	Description	Default	Valid values

Application Healthcheck URL	The path where health check requests are sent to. If this path isn't set, the load balancer attempts to make a TCP connection on port 80 to verify the health status of your application. Set to a path starting with / to send an HTTP GET request to that path. You can also include a protocol (HTTP, HTTPS, TCP, or SSL) and port before the path to check HTTPS connectivity or use a non-default port.	None	Valid values include:
			
			/ (HTTP GET to root path)
			
			`/`*`health`*
			
			`HTTPS:443/`
			
			`HTTPS:443/`*`health`*

Name	Description	Defaul	Valid values
	Note If you use the Elastic Beanstalk console to create an environment, you can't set this option in a configuration file. The console overrides this option with a recommended value.		

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See Recommended values for details.

aws:elasticbeanstalk:application:environment

Configure environment properties for your application.

Namespace: `aws:elasticbeanstalk:application:environment`

Name	Description	Default	Valid values
Any environment variable name.	Pass in key-value pairs.	None	Any environment variable value.

See Environment variables and other software settings for more information.

aws:elasticbeanstalk:application:environmentsecrets

Configure environment variables to serve as *environment secrets* for your application. Environment secrets store AWS Secrets Manager secrets or AWS Systems Manager Parameter Store parameters.

Namespace: `aws:elasticbeanstalk:application:environmentsecrets`

Name	Description	Defaul	Valid values
OptionName	Specifies the name of the environme nt variable to hold the secret store or parameter store value.	None	Any environment variable name.
Value	Specifies the ARN for the value stored in AWS Secrets Manager or AWS Systems Manager Parameter Store. During instance bootstrapping Elastic Beanstalk initiates the environment variable to the value stored in this ARN resource. **Note** Ensure that the necessary permissions are in place for your environment's EC2 instance profile role to access the secret and parameter ARNs. For more information, see Required IAM permissions.	None	Valid ARN value for an AWS Secrets Manager secret or AWS Systems Manager Parameter Store parameter value.

For more information, see Configuring secrets as environment variables.

aws:elasticbeanstalk:cloudwatch:logs

Configure instance log streaming for your application.

Namespace: `aws:elasticbeanstalk:cloudwatch:logs`

Name	Description	Default	Valid values
StreamLogs	Specifies whether to create groups in CloudWatch Logs for proxy and deployment logs, and stream logs from each instance in your environment.	`false`	`true` `false`
DeleteOnTerminate	Specifies whether to delete the log groups when the environment is terminated. If `false`, the logs are kept `RetentionInDays` days.	`false`	`true` `false`
RetentionInDays	The number of days to keep log events before they expire.	7	1, 3, 5, 7, 14, 30, 60, 90, 120, 150, 180, 365, 400, 545, 731, 1827, 3653

aws:elasticbeanstalk:cloudwatch:logs:health

Configure environment health log streaming for your application.

Namespace: `aws:elasticbeanstalk:cloudwatch:logs:health`

Name	Description	Default	Valid values
HealthStreamingEnabled	For environments with enhanced health reporting enabled, specifies whether to create a group in CloudWatch Logs for environment health and archive	`false`	`true` `false`

Name	Description	Default	Valid values
	Elastic Beanstalk environment health data. For information about enabling enhanced health, see `aws:elasticbeanstalk:healthreporting:system`.		
DeleteOnTerminate	Specifies whether to delete the log group when the environment is terminated. If `false`, the health data is kept `RetentionInDays` days.	`false`	`true` `false`
RetentionInDays	The number of days to keep the archived health data before it expires.	7	1, 3, 5, 7, 14, 30, 60, 90, 120, 150, 180, 365, 400, 545, 731, 1827, 3653

aws:elasticbeanstalk:command

Configure the deployment policy for your application code. For more information, see the section called "Deployment options".

Namespace: `aws:elasticbeanstalk:command`

Name	Description	Default	Valid values
DeploymentPolicy	Choose a deployment policy for application version deployments.	`AllAtOnce`	`AllAtOnce`

Name	Description	Default	Valid values
	Note If you use the Elastic Beanstalk console to create an environment, you can't set this option in a configuration file. The console overrides this option with a recommended value.		Rolling RollingWithAdditionalBatch Immutable TrafficSplitting
Timeout	The amount of time, in seconds, to wait for an instance to complete executing commands. Elastic Beanstalk internally adds 240 seconds (four minutes) to the `Timeout` value. For example, the effective timeout by default is 840 seconds (600 + 240), or 14 minutes.	600	1 to 3600
BatchSizeType	The type of number that's specified in **BatchSize**. **Note** If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file. The console and EB CLI override this option with a recommended value.	Percentage	Percentage Fixed

Name	Description	Default	Valid values
BatchSize	The percentage or the fixed number of Amazon EC2 instances in the Auto Scaling group to simultaneously perform deploymen ts on. Valid values vary depending on the **BatchSizeType** setting used. **Note** If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file. The console and EB CLI override this option with a recommended value.	`100`	`1` to `100` (`Percentag e`). `1` to aws:autos caling:as g::MaxSize (`Fixed`)

IgnoreHea lthCheck	Don't cancel a deployment due to failed health checks.	`false`	`true`
			
			`false`

aws:elasticbeanstalk:environment

Configure your environment's architecture and service role.

Namespace: `aws:elasticbeanstalk:environment`

Name	Description	Default	Valid values

Environme ntType	Set to `SingleInstance` to launch one EC2 instance with no load balancer.	`LoadBalan ced`	`SingleIns tance`
			
			`LoadBalan ced`

ServiceRole	The name of an IAM role that Elastic Beanstalk uses to manage resources for the	None	IAM role name, path/

Name	Description	Default	Valid values
	environment. Specify a role name (optionally prefixed with a custom path) or its ARN. Examples: • `aws-elasticbeanstalk-service-role` • *`custom-path`* `/`*`custom-role`* • `arn:aws:iam::123456789012:role/aws-elasticbeanstalk-service-role` **Note** If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file. The console and EB CLI override this option with a recommended value.		name, or ARN

LoadBalancerType	The type of load balancer for your environment. For more information, see the section called “Load balancer”.	`classic`	`classic`
			
			`application`
			
			`network`

Name	Description	Default	Valid values

LoadBalancerIsShared	Specifies whether the environment's load balancer is dedicated or shared. This option can only be set for an Application Load Balancer. It can't be changed after the environment is created.
	
	When `false`, the environment has its own dedicated load balancer, created, and managed by Elastic Beanstalk. When `true`, the environment uses a shared load balancer, created by you and specified in the `SharedLoadBalancer` option of the aws:elbv2:loadbalancer namespace.
		`false`	`true`
			
			`false`

aws:elasticbeanstalk:environment:process:default

Configure your environment's default process.

Namespace: `aws:elasticbeanstalk:environment:process:default`

Name	Description	Default	Valid values
DeregistrationDelay	The amount of time, in seconds, to wait for active requests to complete before deregistering.	`20`	`0` to `3600`
HealthCheckInterval	The interval of time, in seconds, that Elastic Load Balancing checks the health of the Amazon EC2 instances of your application.	With classic or application load balancer: `15` With network load balancer: `30`	With classic or application load balancer: `5` to `300` With network load balancer: `10`, `30`

Name	Description	Default	Valid values
HealthCheckPath	The path that HTTP requests for health checks are sent to.	`/`	A routable path.
HealthCheckTimeout	The amount of time, in seconds, to wait for a response during a health check. This option is only applicable to environments with an application load balancer.	`5`	`1` to `60`
HealthyThresholdCount	The number of consecutive successful requests before Elastic Load Balancing changes the instance health status.	With classic or application load balancer: `3` With network load balancer: `5`	`2` to `10`
MatcherHTTPCode	A comma-separated list of HTTP code(s) that indicate that an instance is healthy. This option is only applicable to environments with a network or application load balancer.	`200`	With application load balancer: `200` to `499` With network load balancer: `200` to `399`
Port	Port that the process listens on.	`80`	`1` to `65535`

Name	Description	Default	Valid values
Protocol	The protocol that the process uses. With an applicati on load balancer, you can only set this option to HTTP or HTTPS. With a network load balancer, you can only set this option to TCP.	With classic or application load balancer: HTTP With network load balancer: TCP	TCP HTTP HTTPS

StickinessEnabled	Set to true to enable sticky sessions.
	
	This option is only applicable to environments with an application load balancer.
		`'false'`	`'false'`
			
			`'true'`

StickinessLBCookie Duration	The lifetime, in seconds, of the sticky session cookie. This option is only applicable to environments with an application load balancer.	`86400` (one day)	`1` to `604800`

Name	Description	Default	Valid values
StickinessType	Set to `lb_cookie` to use cookies for sticky sessions. This option is only applicable to environments with an application load balancer.	`lb_cookie`	`lb_cookie`
UnhealthyThreshold Count	The number of consecutive unsuccessful requests before Elastic Load Balancing changes the instance health status.	`5`	`2` to `10`

aws:elasticbeanstalk:environment:process:process_name

Configure additional processes for your environment.

Namespace: `aws:elasticbeanstalk:environment:process:`*`process_name`*

Name	Description	Default	Valid values
DeregistrationDelay	The amount of time, in seconds, to wait for active requests to complete before deregistering.	`20`	`0` to `3600`
HealthCheckInterval	The interval, in seconds, that Elastic Load Balancing checks the health	With classic or application load balancer: `15`	With classic or application load balancer: `5` to `300`

Name	Description	Default	Valid values
	of Amazon EC2 instances for your application.	With network load balancer: `30`	With network load balancer: `10`, `30`
HealthCheckPath	The path that HTTP requests for health checks are sent to.	`/`	A routable path.
HealthCheckTimeout	The amount of time, in seconds, to wait for a response during a health check. This option is only applicable to environments with an application load balancer.	`5`	`1` to `60`
HealthyThresholdCount	The number of consecutive successful requests before Elastic Load Balancing changes the instance health status.	With classic or application load balancer: `3` With network load balancer: `5`	`2` to `10`

Name	Description	Default	Valid values
MatcherHTTPCode	A comma-separated list of HTTP code(s) that indicates that an instance is healthy. This option is only applicable to environments with a network or applicati on load balancer.	`200`	With application load balancer: `200` to `499` With network load balancer: `200` to `399`
Port	The port that the process listens on.	`80`	`1` to `65535`

Protocol	The protocol that the process uses.
	
	With an applicati on load balancer, you can only set this option to `HTTP` or `HTTPS`.
	
	With a network load balancer, you can only set this option to `TCP`.
		With classic or application load balancer: `HTTP`
		
		With network load balancer: `TCP`
			`TCP`
			
			`HTTP`
			
			`HTTPS`

StickinessEnabled	Set to true to enable sticky sessions.
	
	This option is only applicable to environments with an application load balancer.
		`'false'`	`'false'`
			
			`'true'`

Name	Description	Default	Valid values
StickinessLBCookie Duration	The lifetime, in seconds, of the sticky session cookie. This option is only applicable to environments with an application load balancer.	`86400` (one day)	`1` to `604800`
StickinessType	Set to `lb_cookie` to use cookies for sticky sessions. This option is only applicable to environments with an application load balancer.	`lb_cookie`	`lb_cookie`
UnhealthyThreshold Count	The number of consecutive unsuccessful requests before Elastic Load Balancing changes the instance health status.	`5`	`2` to `10`

aws:elasticbeanstalk:environment:proxy:staticfiles

You can use the following namespace to configure the proxy server to serve static files. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application. This reduces the number of requests that your application has to process.

Map a path served by the proxy server to a folder in your source code that contains static assets. Each option that you define in this namespace maps a different path.

Note

This namespace applies to platform branches based on Amazon Linux 2 and later. If your environment uses a platform version based on Amazon Linux AMI (preceding Amazon Linux 2), refer to the section called "Platform specific options" for platform-specific static file namespaces.

Namespace: `aws:elasticbeanstalk:environment:proxy:staticfiles`

Name	Value
The path where the proxy server serves the files. Start the value with `/`. For example, specify `/images` to serve files at *`subdomain`* `.eleasticbeanstalk.com/images` .	The name of the folder containing the files. For example, specify `staticimages` to serve files from a folder named `staticimages` at the top level of your source bundle.

aws:elasticbeanstalk:healthreporting:system

Configure enhanced health reporting for your environment.

Namespace: `aws:elasticbeanstalk:healthreporting:system`

Name	Description	Default	Valid values

SystemType	The health reporting system (basic or enhanced). Enhanced health reporting requires a service role and a version 2 or newer platform version.	`basic`	`basic`
			
			`enhanced`

Name	Description	Default	Valid values
	Note If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file. The console and EB CLI override this option with a recommended value.		
ConfigDocument	A JSON document that describes the environment and instance metrics to publish to CloudWatch.	None	

EnhancedHealthAuthEnabled	Enables authorization for the internal API that Elastic Beanstalk uses to communicate enhanced health information from your environment instances to the Elastic Beanstalk service.
	
	For more information, see the section called "Enhanced health roles".
	
	Note
	This option is only applicable to enhanced health reporting (such as when `SystemType` is set to `enhanced`).
		`true`	`true`
			
			`false`

Name	Description	Default	Valid values
HealthCheckSuccessThreshold	Lowers the threshold for instances to pass health checks. **Note** If you use the Elastic Beanstalk console to create an environment, you can't set this option in a configuration file. The console overrides this option with a recommended value.	`Ok`	`Ok` `Warning` `Degraded` `Severe`

aws:elasticbeanstalk:hostmanager

Configure the EC2 instances in your environment to upload rotated logs to Amazon S3.

Namespace: `aws:elasticbeanstalk:hostmanager`

Name	Description	Default	Valid values
LogPublicationControl	Copy the log files of the Amazon EC2 instances for your application to the Amazon S3 bucket that's associated with your application.	`false`	`true` `false`

aws:elasticbeanstalk:managedactions

Configure managed platform updates for your environment.

Namespace: `aws:elasticbeanstalk:managedactions`

Name	Description	Default	Valid values
ManagedActionsEnabled	Enable managed platform updates.	`false`	`true` `false`

Name	Description	Default	Valid values
	When you set this to `true`, you must also specify a `PreferredStartTime` and `UpdateLevel`.		
PreferredStartTime	Configure a maintenance window for managed actions in UTC. For example, `"Tue:09:00"`.	None	Day and time in the *`day`*:*`hour`*:*`minute`* format.
ServiceRoleForManagedUpdates	The name of an IAM role that Elastic Beanstalk uses to perform managed platform updates for your environment. You can use either the same role that you specified for the `ServiceRole` option of the `aws:elasticbeanstalk:environment` namespace, or your account's managed updates service-linked role. In the latter case, if the account doesn't have a managed-updates service-linked role yet, Elastic Beanstalk creates it.	None	Same as `ServiceRole` or `AWSServiceRoleForElasticBeanstalkManagedUpdates`

aws:elasticbeanstalk:managedactions:platformupdate

Configure managed platform updates for your environment.

Namespace: `aws:elasticbeanstalk:managedactions:platformupdate`

Name	Description	Default	Valid values
UpdateLevel	The highest level of update to apply with managed platform updates. Platforms are versioned *`major`*.*`minor`*.*`patch`*. For example, 2.0.8 has a major version of 2, a minor version of 0, and a patch version of 8.	None	`patch` for patch version updates only. `minor` for both minor and patch version updates.

InstanceRefreshEnabled	Enable weekly instance replacement.
	
	This requires `ManagedActionsEnabled` to be set to `true`.
		`false`	`true`
			
			`false`

aws:elasticbeanstalk:monitoring

Configure your environment to terminate EC2 instances that fail health checks.

Namespace: `aws:elasticbeanstalk:monitoring`

Name	Description	Default	Valid values

Automatically Terminate Unhealthy Instances	Terminate an instance if it fails health checks.
	
	Note
	This option was only supported on legacy environments. It determined the health of an instance based on
		`true`	`true`
			
			`false`

Name	Description	Default	Valid values
	being able to reach it and on other instance-based metrics. Elastic Beanstalk doesn't provide a way to automatically terminate instances based on application health.		

aws:elasticbeanstalk:sns:topics

Configure notifications for your environment.

Namespace: `aws:elasticbeanstalk:sns:topics`

Name	Description	Default	Valid values
Notification Endpoint	The endpoint where you want to be notified of important events affecting your application. **Note** If you use the Elastic Beanstalk console to create an environment, you can't set this option in a configuration file. The console overrides this option with a recommended value.	None	
Notification Protocol	The protocol that's used to send notifications to your endpoint.	`email`	`http`

Name	Description	Default	Valid values

			`https`
			
			`email`
			
			`email-json`
			
			`sqs`

Notification Topic ARN	The Amazon Resource Name (ARN) for the topic you subscribed to.	None	
Notification Topic Name	The name of the topic you subscribed to.	None	

aws:elasticbeanstalk:sqsd

Configure the Amazon SQS queue for a worker environment.

Namespace: `aws:elasticbeanstalk:sqsd`

Name	Description	Default	Valid values
WorkerQue ueURL	The URL of the queue that the daemon in the worker environme nt tier reads messages from. **Note** When you don't specify a value, the queue that Elastic Beanstalk automatically creates is a standard Amazon SQS queue. When you provide a value, you can provide the URL of either	automatic ally generated	If you don't specify a value, then Elastic Beanstalk automatically creates a queue.

Name	Description	Default	Valid values
	a standard or a FIFO Amazon SQS queue. Be aware that if you provide a FIFO queue, periodic tasks aren't supported.		
HttpPath	The relative path to the application that HTTP POST messages are sent to.	/	

MimeType	The MIME type of the message that's sent in the HTTP POST request.	`application/json`	`application/json`
			
			`application/x-www-form-urlencoded`
			
			`application/xml`
			
			`text/plain`
			
			Custom MIME type.

HttpConnections	The maximum number of concurrent connections to any applications that are within an Amazon EC2 instance. **Note** If you use the Elastic Beanstalk console to create an environment, you can't set this option in a configuration file. The console overrides this option with a recommended value.	`50`	`1` to `100`

Name	Description	Default	Valid values
ConnectTimeout	The amount of time, in seconds, to wait for successful connections to an application.	5	`1` to `60`
InactivityTimeout	The amount of time, in seconds, to wait for a response on an existing connection to an application. The message is reprocessed until the daemon receives a 200 (OK) response from the application in the worker environment tier or the `RetentionPeriod` expires.	299	`1` to `36000`
VisibilityTimeout	The amount of time, in seconds, an incoming message from the Amazon SQS queue is locked for processing. After the configured amount of time has passed, then the message is again made visible in the queue for any other daemon to read.	300	`0` to `43200`
ErrorVisibilityTimeout	The amount of time, in seconds, that elapses before Elastic Beanstalk returns a message to the Amazon SQS queue after a processing attempt fails with an explicit error.	2 seconds	`0` to `43200` seconds
RetentionPeriod	The amount of time, in seconds, a message is valid and is actively processed for.	`345600`	`60` to `1209600`

Name	Description	Default	Valid values
MaxRetries	The maximum number of attempts that Elastic Beanstalk attempts to send the message to the web application that will process it before moving the message to the dead-letter queue.	`10`	`1` to `100`

aws:elasticbeanstalk:trafficsplitting

Configure traffic-splitting deployments for your environment.

This namespace applies when you set the `DeploymentPolicy` option of the aws:elasticbeanstalk:command namespace to `TrafficSplitting`. For more information about deployment policies, see the section called "Deployment options".

Namespace: `aws:elasticbeanstalk:trafficsplitting`

Name	Description	Default	Valid values
NewVersionPercent	The initial percentage of incoming client traffic that Elastic Beanstalk shifts to environment instances running the new application version you're deploying.	`10`	`1` to `100`
EvaluationTime	The time period, in minutes, that Elastic Beanstalk waits after an initial healthy deployment before proceeding to shift all incoming client traffic to the new application version that you're deploying.	`5`	`3` to `600`

aws:elasticbeanstalk:xray

Run the AWS X-Ray daemon to relay trace information from your X-Ray integrated application.

Namespace: `aws:elasticbeanstalk:xray`

Name	Description	Default	Valid values
`XRayEnabled`	Set to `true` to run the X-Ray daemon on the instances in your environment.	`false`	`true` `false`

aws:elb:healthcheck

Configure healthchecks for a Classic Load Balancer.

Namespace: `aws:elb:healthcheck`

Name	Description	Default	Valid values
HealthyThreshold	The number of consecutive successful requests before Elastic Load Balancing changes the instance health status.	3	`2` to `10`
Interval	The interval that Elastic Load Balancing checks the health of your application's Amazon EC2 instances at.	`10`	`5` to `300`
Timeout	The amount of time, in seconds, that Elastic Load Balancing waits for a response before it considers the instance nonresponsive.	5	`2` to `60`
Unhealthy Threshold	The number of consecutive unsuccessful requests before Elastic Load Balancing changes the instance health status.	5	`2` to `10`
(deprecated) Target	The destination on a backend instance that health checks are sent to. Use `Application Healthcheck URL` in the [`aws:elasticbeanstalk:application`](#) namespace instead.	`TCP:80`	Target in the format *PROTOCOL*:*PORT*/*PATH*

aws:elb:loadbalancer

Configure your environment's Classic Load Balancer.

Several of the options in this namespace are no longer supported in favor of listener-specific options in the aws:elb:listener namespace. With these options that aren't supported anymore, you can only configure two listeners (one secure and one unsecure) on standard ports.

Namespace: `aws:elb:loadbalancer`

Name	Description	Default	Valid values

CrossZone	Configure the load balancer to route traffic evenly across all instances in all Availability Zones rather than only within each zone.
	
	Note
	If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file. The console and EB CLI override this option with a recommended value.
		`false`	`true`
			
			`false`

SecurityGroups	Assign one or more security groups that you created to the load balancer. Required if `DisableDefaultEC2SecurityGroup` (aws:autoscaling:launchconfiguration) is set to `true`. Load balanced environments that have opted out of the default Elastic Beanstalk EC2 security group must provide one or more security groups with this option. For more information, see Managing EC2 security groups.	None	One or more security group IDs.

Name	Description	Default	Valid values
ManagedSecurityGroup	Assign an existing security group to the load balancer for your environment, instead of creating a new one. To use this setting, update the `SecurityGroups` setting in this namespace to include your security group's ID, and remove the ID of the security group that was created automatically, if one was created. To allow traffic from the load balancer to your environment's EC2 instances, Elastic Beanstalk adds a rule to the security group of the instances that allows inbound traffic from the managed security group.	None	A security group ID.

(deprecated) LoadBalancerHTTPPort	The port to listen on for the unsecure listener.	`80`	`OFF`
			
			`80`

(deprecated) LoadBalancerPortProtocol	The protocol to use on the unsecure listener.	`HTTP`	`HTTP`
			
			`TCP`

(deprecated) LoadBalancerHTTPSPort	The port to listen on for the secure listener.	`OFF`	`OFF`
			
			`443`
			
			`8443`

(deprecated) LoadBalancerSSLPortProtocol	The protocol to use on the secure listener.	`HTTPS`	`HTTPS`
			
			`SSL`

(deprecated) SSLCertificateId	The Amazon Resource Name (ARN) of an SSL certificate to bind to the secure listener.	None	

aws:elb:listener

Configure the default listener (port 80) on a Classic Load Balancer.

Namespace: `aws:elb:listener`

Name	Description	Default	Valid values
ListenerProtocol	The protocol used by the listener.	HTTP	HTTP TCP
InstancePort	The port that this listener uses to communicate with the EC2 instances.	80	1 to 65535
InstanceProtocol	The protocol that this listener uses to communicate with the EC2 instances. It must be at the same internet protocol layer as the `ListenerProtocol` . It also must have the same security level as any other listener using the same `InstancePort` as this listener. For example, if `ListenerProtocol` is HTTPS (application layer, using a secure connection), you can set `InstanceProtocol` to HTTP (also at the application layer, using an insecure connection). If, in addition, you set `InstancePort` to `80`, you must set `InstanceProtocol` to HTTP in all other listeners with `InstancePort` set to `80`.	HTTP when `ListenerProtocol` is HTTP TCP when `ListenerProtocol` is TCP	HTTP or HTTPS when `ListenerProtocol` is HTTP or HTTPS TCP or SSL when `ListenerProtocol` is TCP or SSL
PolicyNames	A comma-separated list of policy names to apply to the port for this listener. We recommend that you use the LoadBalancerPorts option of the aws:elb:policies namespace instead.	None	

Name	Description	Default	Valid values
ListenerEnabled	Specifies whether this listener is enabled. If you specify `false`, the listener isn't included in the load balancer.	`true`	`true` `false`

aws:elb:listener:listener_port

Configure additional listeners on a Classic Load Balancer.

Namespace: `aws:elb:listener:`*`listener_port`*

Name	Description	Default	Valid values
ListenerProtocol	The protocol used by the listener.	HTTP	HTTP HTTPS TCP SSL
InstancePort	The port that this listener uses to communicate with the EC2 instances.	The same as *`listener_port`* .	1 to 65535
InstanceProtocol	The protocol that this listener uses to communicate with the EC2 instances. It must be at the same internet protocol layer as the `ListenerProtocol` . It also must have the same security level as any other listener using the same `InstancePort` as this listener. For example, if `ListenerProtocol` is HTTPS (application layer, using a secure connection), you can set `InstanceProtocol` to HTTP (also at the applicati	HTTP when `ListenerProtocol` is HTTP or HTTPS TCP when `ListenerProtocol` is TCP or SSL	HTTP or HTTPS when `ListenerProtocol` is HTTP or HTTPS TCP or SSL when `ListenerProtocol`

Name	Description	Default	Valid values
	on layer, using an insecure connection). If, in addition, you set `InstancePort` to `80`, you must set `InstanceProtocol` to `HTTP` in all other listeners with `InstancePort` set to `80`.		is TCP or `SSL`
PolicyNames	A comma-separated list of policy names to apply to the port for this listener. We suggest that you use the LoadBalancerPorts option of the aws:elb:policies namespace instead.	None	
SSLCertificateId	The Amazon Resource Name (ARN) of an SSL certificate to bind to the listener.	None	
ListenerEnabled	Specifies whether this listener is enabled. If you specify `false`, the listener isn't included in the load balancer.	`true` if any other option is set. `false` otherwise.	`true` `false`

aws:elb:policies

Modify the default stickiness and global load balancer policies for a Classic Load Balancer.

Namespace: `aws:elb:policies`

Name	Description	Default	Valid values
ConnectionDrainingEnabled	Specifies whether the load balancer maintains existing connections to instances that have become unhealthy or deregistered to complete in-progress requests.	`false`	`true` `false`

Name	Description	Default	Valid values
	Note If you use the Elastic Beanstalk console or EB CLI to create an environment, you can't set this option in a configuration file. The console and EB CLI override this option with a recommended value.		
ConnectionDraining Timeout	The maximum number of seconds that the load balancer maintains existing connections to an instance during connection draining before forcibly closing the connections. **Note** If you use the Elastic Beanstalk console to create an environme nt, you can't set this option in a configuration file. The console overrides this option with a recommended value.	`20`	`1` to `3600`
ConnectionSettingI dleTimeout	The amount of time, in seconds, that the load balancer waits for any data to be sent or received over the connection. If no data has been sent or received after this time period elapses, the load balancer closes the connection.	`60`	`1` to `3600`

Name	Description	Default	Valid values
LoadBalancerPorts	A comma-separated list of the listener ports that the default policy (`AWSEB-ELB-StickinessPolicy`) applies to.	None	You can use `:all` to indicate all listener ports
Stickiness Cookie Expiration	The amount of time, in seconds, that each cookie is valid. Uses the default policy (`AWSEB-ELB-StickinessPolicy`).	`0`	`0` to `1000000`
Stickiness Policy	Binds a user's session to a specific server instance so that all requests coming from the user during the session are sent to the same server instance. Uses the default policy (`AWSEB-ELB-StickinessPolicy`).	`false`	`true` `false`

aws:elb:policies:policy_name

Create additional load balancer policies for a Classic Load Balancer.

Namespace: `aws:elb:policies:`*`policy_name`*

Name	Description	Default	Valid values
CookieName	The name of the application-generated cookie that controls the session lifetimes of a `AppCookieStickinessPolicyType` policy. This policy can be associated only with HTTP/HTTPS listeners.	None	
InstancePorts	A comma-separated list of the instance ports that this policy applies to.	None	A list of ports, or `:all`
LoadBalancerPorts	A comma-separated list of the listener ports that this policy applies to.	None	A list of ports, or `:all`

Name	Description	Default	Valid values
ProxyProtocol	For a `ProxyProtocolPolicyType` policy, specifies whether to include the IP address and port of the originating request for TCP messages. This policy can be associated only with TCP/SSL listeners.	None	`true` `false`
PublicKey	The contents of a public key for a `PublicKeyPolicyType` policy to use when authenticating the backend server or servers. This policy can't be applied directly to backend servers or listeners. It must be part of a `BackendServerAuthenticationPolicyType` policy.	None	
PublicKeyPolicyNames	A comma-separated list of policy names (from the `PublicKeyPolicyType` policies) for a `BackendServerAuthenticationPolicyType` policy that controls authentication to a backend server or servers. This policy can be associated only with backend servers that are using HTTPS/SSL.	None	
SSLProtocols	A comma-separated list of SSL protocols to be enabled for a `SSLNegotiationPolicyType` policy that defines the ciphers and protocols that are accepted by the load balancer. This policy can be associated only with HTTPS/SSL listeners.	None	

Name	Description	Default	Valid values
SSLReferencePolicy	The name of a predefined security policy that adheres to AWS security best practices and that you want to activate for a `SSLNegotiationPolicyType` policy that defines the ciphers and protocols that are accepted by the load balancer. This policy can be associated only with HTTPS/SSL listeners.	None	
Stickiness Cookie Expiration	The amount of time, in seconds, that each cookie is valid.	`0`	`0` to `1000000`
Stickiness Policy	Binds a user's session to a specific server instance so that all requests coming from the user during the session are sent to the same server instance.	`false`	`true` `false`

aws:elbv2:listener:default

Configure the default listener (port 80) on an Application Load Balancer or a Network Load Balancer.

This namespace doesn't apply to an environment that uses a shared load balancer. Shared load balancers don't have a default listener.

Namespace: `aws:elbv2:listener:default`

Name	Description	Default	Valid values
DefaultProcess	The name of the process to forward traffic to when no rules match.	`default`	A process name.

ListenerEnabled	Set to `false` to disable the listener.	`true`	`true`
			
			`false`

Name	Description	Default	Valid values
	You can use this option to disable the default listener on port 80.		
Protocol	The protocol of traffic to process.	With application load balancer: HTTP With network load balancer: TCP	With application load balancer: HTTP, HTTPS With network load balancer: TCP
Rules	A list of rules to apply to the listener This option is only applicable to environments with an Application Load Balancer.	None	A comma-separated list of rule names.
SSLCertificateArns	The Amazon Resource Name (ARN) of the SSL certifica te to bind to the listener. This option is only applicable to environments with an Application Load Balancer.	None	The ARN of a certifica te stored in IAM or ACM.

Name	Description	Default	Valid values
SSLPolicy	Specify a security policy to apply to the listener. This option is only applicable to environments with an Application Load Balancer.	None (ELB default)	The name of a load balancer security policy.

aws:elbv2:listener:listener_port

Configure additional listeners on an Application Load Balancer or a Network Load Balancer.

Note

For a shared Application Load Balancer, you can specify only the `Rule` option. The other options aren't applicable to shared load balancers.

Namespace: `aws:elbv2:listener:`*`listener_port`*

Name	Description	Default	Valid values
DefaultProcess	The name of the process where traffic is forwarded when no rules match.	`default`	A process name.

ListenerEnabled	Set to `false` to disable the listener. You can use this option to disable the default listener on port 80.	`true`	`true`
			
			`false`

Name	Description	Default	Valid values
Protocol	The protocol of traffic to process.	With application load balancer: HTTP With network load balancer: TCP	With application load balancer: HTTP, HTTPS With network load balancer: TCP
Rules	List of rules to apply to the listener This option is applicable only to environments with an Application Load Balancer. If your environme nt uses a shared Application Load Balancer, and you don't specify this option for any listener, Elastic Beanstalk automatic ally associates the `default` rule with a port 80 listener.	None	A comma-separated list of rule names.

Name	Description	Default	Valid values
SSLCertificateArns	The Amazon Resource Name (ARN) of the SSL certificate to bind to the listener. This option is only applicable to environments with an Application Load Balancer.	None	The ARN of a certificate stored in IAM or ACM.
SSLPolicy	Specify a security policy to apply to the listener. This option is only applicable to environments with an Application Load Balancer.	None (ELB default)	The name of a load balancer security policy.

aws:elbv2:listenerrule:rule_name

Define listener rules for an Application Load Balancer. If a request matches the host names or paths in a rule, the load balancer forwards it to the specified process. To use a rule, add it to a listener with the `Rules` option in the `aws:elbv2:listener:`*`listener_port`* namespace.

Note

This namespace isn't applicable to environments with a network load balancer.

Namespace: `aws:elbv2:listenerrule:`*`rule_name`*

Name	Description	Default	Valid values
HostHeaders	A list of host names to match. For example, `my.example.com` .	Dedicated load balancer: None Shared load balancer: The environment's CNAME	Each name can contain up to 128 characters. A pattern can include both uppercase and lowercase letters, numbers, hyphens (–), and up to three wildcard characters (* matches zero or more characters; ? matches exactly one character). You can list more than one name, each separated by a comma. Application Load Balancer supports up to five combined `HostHeader` and `PathPattern` rules. For more information, see Host conditions in the *User Guide for Application Load Balancers*.
PathPatterns	The path patterns to match (for example, `/img/*`). This option is only applicable to environments with an application load balancer.	None	Each pattern can contain up to 128 characters. A pattern can include uppercase and lowercase letters,

Name	Description	Default	Valid values
			numbers, hyphens (–), and up to three wildcard characters (* matches zero or more characters; ? matches exactly one character). You can add multiple comma-separated path patterns. Application Load Balancer supports up to five combined `HostHeader` and `PathPattern` rules. For more information, see [Path conditions](#) in the *User Guide for Application Load Balancers*.
Priority	The precedence of this rule when multiple rules match. The lower number takes precedence. No two rules can have the same priority. With a shared load balancer, Elastic Beanstalk treats rule priorities as relative across sharing environments, and maps them to absolute priorities during creation.	`1`	`1` to `1000`
Process	The name of the [process](#) to forward traffic when this rule matches the request.	`default`	A process name.

aws:elbv2:loadbalancer

Configure an Application Load Balancer.

For a shared load balancer, only the `SharedLoadBalancer` and `SecurityGroups` options are valid.

Note

This namespace isn't applicable to environments with a Network Load Balancer.

Namespace: `aws:elbv2:loadbalancer`

Name	Description	Default	Valid values
AccessLogsS3Bucket	The Amazon S3 bucket where access logs are stored. The bucket must be in the same Region as the environment and allow the load balancer write access.	None	A bucket name.
AccessLogsS3Enabled	Enable access log storage.	`false`	`true` `false`
AccessLogsS3Prefix	A prefix to prepend to access log names. By default, the load balancer uploads logs to a directory named AWSLogs in the bucket you specify. Specify a prefix to place the AWSLogs directory inside another directory.	None	
IdleTimeout	The amount of time, in seconds, to wait for a request to complete before closing connections to client and instance.	None	`1` to `3600`

Name	Description	Default	Valid values
ManagedSecurityGroup	Assign an existing security group to your environment's load balancer, instead of creating a new one. To use this setting, update the `SecurityGroups` setting in this namespace to include your security group's ID, and remove the automatically created security group's ID, if one exists. To allow traffic from the load balancer to the EC2 instances for your environment, Elastic Beanstalk adds a rule to the security group of your instances that allows inbound traffic from the managed security group.	The security group that Elastic Beanstalk creates for your load balancer	A security group ID.

Name	Description	Default	Valid values
SecurityGroups	A list of security groups to attach to the load balancer. Required if `DisableDefaultEC2SecurityGroup` (aws:autoscaling:launchconfiguration) is set to `true`. Load balanced environments that have opted out of the default Elastic Beanstalk EC2 security group must provide one or more security groups with this option. For more information, see Managing EC2 security groups. For a shared load balancer, if you don't specify this value, Elastic Beanstalk checks if an existing security group that it manages is already attached to the load balancer. If one isn't attached to the load balancer, Elastic Beanstalk creates a security group and attaches it to the load balancer. Elastic Beanstalk deletes this security group when the last environment sharing the load balancer terminates. The load balancer security groups are used to set up the Amazon EC2 instance security group ingress rule.	The security group that Elastic Beanstalk creates for your load balancer	Comma-separated list of security group IDs.

Name	Description	Default	Valid values
SharedLoadBalancer	The Amazon Resource Name (ARN) of a shared load balancer. This option is relevant only to an Application Load Balancer. It's required when the `LoadBalancerIsShared` option of the [aws:elasticbeanstalk:environment](#) namespace is set to `true`. You can't change the shared load balancer ARN after the environment is created. Criteria for a valid value: • It must be a valid, active load balancer in the AWS Region where the environment is located. • It must be in the same Amazon Virtual Private Cloud (Amazon VPC) as the environment. • It can't be a load balancer that was created by Elastic Beanstalk as the dedicated load balancer for another environment. You can identify these dedicated load balancers by using the prefix `awseb-`. Example: `arn:aws:elasticloadbalancing:us-east-2:123456789012:lo`	None	ARN of a valid load balancer that meets all of the criteria described here.

Name	Description	Default	Valid values
	`adbalancer/app/FrontEndLB/0dbf78d8ad96abbc`		

aws:rds:dbinstance

Configure an attached Amazon RDS DB instance.

Namespace: `aws:rds:dbinstance`

Name	Description	Default	Valid values
DBAllocatedStorage	The allocated database storage size, specified in gigabytes.	MySQL: `5` Oracle: `10` sqlserver-se: `200` sqlserver-ex: `30` sqlserver-web: `30`	MySQL: `5`-`1024` Oracle: `10`-`1024` sqlserver: cannot be modified

DBDeletionPolicy	Specifies whether to retain, delete, or create snapshot of the DB instance when an environment is terminated.
	
	This option works in conjunction with `HasCoupledDatabase` , also an option of this namespace.
	
	Warning
	Deleting a DB instance results in permanent data loss.
		`Delete`	`Delete`
			
			`Retain`
			
			`Snapshot`

Name	Description	Default	Valid values

DBEngine	The name of the database engine to use for this instance.	`mysql`	`mysql`
			
			`oracle-se1`
			
			`sqlserver-ex`
			
			`sqlserver-web`
			
			`sqlserver-se`
			
			`postgres`

DBEngineVersion	The version number of the database engine.	`5.5`	
DBInstanceClass	The database instance type.	`db.t2.micro` (`db.m1.large` for an environment not running in an Amazon VPC)	For more information, see DB Instance Class in the *Amazon Relational Database Service User Guide*.
DBPassword	The name of master user password for the database instance.	None	
DBSnapshotIdentifier	The identifier for the DB snapshot to restore from.	None	
DBUser	The name of master user for the DB Instance.	**ebroot**	

Name	Description	Default	Valid values

HasCoupledDatabase	Specifies whether a DB instance is coupled to your environment. If toggled to `true`, Elastic Beanstalk creates a new DB instance coupled to your environment. If toggled to `false`, Elastic Beanstalk initiates decoupling of the DB instance from your environment.
	
	This option works in conjunction with `DBDeletionPolicy` , also an option of this namespace.
	
	Note
	Note: If you toggle this value back to `true` after decoupling the previous database, Elastic Beanstalk creates a new database with the previous database option settings. However, to maintain the security of your environment, it doesn't retain the existing `DBUser` and `DBPassword` settings. You need to specify `DBUser` and `DBPassword` again.
		`false`	`true`
			
			`false`

MultiAZDatabase	Specifies whether a database instance Multi-AZ deployment needs to be created. For more information about Multi-AZ deployments with Amazon Relational Database Service (RDS), see Regions and Availability Zones in the *Amazon Relational Database Service User Guide*.	`false`	`true`
			
			`false`

Platform specific options

Some Elastic Beanstalk platforms define option namespaces that are specific to the platform. These namespaces and their options are listed below for each platform.

Note

Previously, in platform versions based on Amazon Linux AMI (preceding Amazon Linux 2), the following two features and their respective namespaces were considered to be platform-specific features, and were listed here per platform:

- **Proxy configuration for static files** – `aws:elasticbeanstalk:environment:proxy:staticfiles`
- **AWS X-Ray support** – `aws:elasticbeanstalk:xray`

In Amazon Linux 2 platform versions, Elastic Beanstalk implements these features in a consistent way across all supporting platforms. The related namespace are now listed in the the section called "General options" page. We only kept mention of them on this page for platforms who had differently-named namespaces.

Platforms

- Docker platform options
- Go platform options
- Java SE platform options
- Java with Tomcat platform options
- .NET Core on Linux platform options
- .NET platform options
- Node.js platform options
- PHP platform options
- Python platform options
- Ruby platform options

Docker platform options

The following Docker-specific configuration options apply to the Docker and Preconfigured Docker platforms.

Note

These configuration options do not apply to

- Docker platform (Amazon Linux 2) with Docker Compose
- Multicontainer Docker platform (Amazon Linux AMI AL1) - this platform is retired

Namespace: `aws:elasticbeanstalk:environment:proxy`

Name	Description	Default	Valid values
ProxyServer	Specifies the web server to use as a proxy.	`nginx`	`nginx` none – *Amazon Linux AM* and *Docker w/DC only*

Go platform options

Amazon Linux AMI (pre-Amazon Linux 2) platform options

Namespace: `aws:elasticbeanstalk:container:golang:staticfiles`

You can use the following namespace to configure the proxy server to serve static files. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application. This reduces the number of requests that your application has to process.

Map a path served by the proxy server to a folder in your source code that contains static assets. Each option that you define in this namespace maps a different path.

Name	Value
Path where the proxy server will serve the files. Example: `/images` to serve files at *`subdomain`* `.eleasticbeanstalk.com/images` .	Name of the folder containing the files. Example: `staticimages` to serve files from a folder named `staticimages` at the top level of your source bundle.

Java SE platform options

Amazon Linux AMI (pre-Amazon Linux 2) platform options

Namespace: `aws:elasticbeanstalk:container:java:staticfiles`

You can use the following namespace to configure the proxy server to serve static files. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application. This reduces the number of requests that your application has to process.

Map a path served by the proxy server to a folder in your source code that contains static assets. Each option that you define in this namespace maps a different path.

Name	Value
Path where the proxy server will serve the files. Example: `/images` to serve files at *`subdomain`* `.eleasticbeanstalk.com/images` .	Name of the folder containing the files. Example: `staticimages` to serve files from a folder named `staticimages` at the top level of your source bundle.

Java with Tomcat platform options

Namespace: `aws:elasticbeanstalk:application:environment`

Name	Description	Default	Valid values
JDBC_CONNECTION_STRING	The connection string to an external database.	n/a	n/a

See Environment variables and other software settings for more information.

Namespace: `aws:elasticbeanstalk:container:tomcat:jvmoptions`

Name	Description	Default	Valid values
JVM Options	Pass command-line options to the JVM at startup.	n/a	n/a
Xmx	Maximum JVM heap sizes.	`256m`	n/a
XX:MaxPermSize	Section of the JVM heap that is used to store class definitions and associated metadata. **Note** This option only applies to Java versions earlier than Java 8, and isn't supported on Elastic Beanstalk Tomcat platforms based on Amazon Linux 2 and later.	`64m`	n/a
Xms	Initial JVM heap sizes.	`256m`	n/a
`optionName`	Specify arbitrary JVM options in addition to the those defined by the Tomcat platform.	n/a	n/a

Namespace: `aws:elasticbeanstalk:environment:proxy`

Name	Description	Default	Valid values

GzipCompression	Set to `false` to disable response compression.
	
	Only valid on Amazon Linux AMI (preceding Amazon Linux 2) platform versions.
		`true`	`true`
			
			`false`

ProxyServer	Set the proxy to use on your environment's instances. If you set this option to `apache`, Elastic Beanstalk uses Apache 2.4.
	
	Set to `apache/2.2` if your application isn't ready to migrate away from Apache 2.2 due to incompatible proxy configuration settings. *This value is only valid on Amazon Linux AMI (preceding Amazon Linux 2) platform versions.*
	
	Set to `nginx` to use nginx. This is the default starting with Amazon Linux 2 platform versions.
	
	For more information, see Configuring the proxy server.
		`nginx` (Amazon Linux 2)
		
		`apache` (Amazon Linux AMI)
			`apache`
			
			`apache/2.2` – *Amazon Linux AMI only*
			
			`nginx`

.NET Core on Linux platform options

Namespace: `aws:elasticbeanstalk:environment:proxy`

Name	Description	Default	Valid values
ProxyServer	Specifies the web server to use as a proxy.	`nginx`	`nginx` none

.NET platform options

Namespace: `aws:elasticbeanstalk:container:dotnet:apppool`

Name	Description	Default	Valid values

Target Runtime	Choose the version of .NET Framework for your application.	`4.0`	`2.0`
			
			`4.0`

Enable 32-bit Applications	Set to `True` to run 32-bit applications.	`False`	`True`
			
			`False`

Node.js platform options

Namespace: `aws:elasticbeanstalk:environment:proxy`

Name	Description	Default	Valid values

ProxyServer	Set the proxy to use on your environment's instances.	`nginx`	`apache`
			
			`nginx`

Amazon Linux AMI (pre-Amazon Linux 2) platform options

Namespace: `aws:elasticbeanstalk:container:nodejs`

Name	Description	Default	Valid values
NodeCommand	Command used to start the Node.js application. If an empty string is specified, `app.js` is used, then `server.js` , then `npm start` in that order.	""	n/a
NodeVersion	Version of Node.js. For example, `4.4.6` Supported Node.js versions vary between Node.js platform versions. See Node.js	varies	varies

Name	Description	Default	Valid values
	in the *AWS Elastic Beanstalk Platforms* document for a list of the currently supported versions. **Note** When support for the version of Node.js that you are using is removed from the platform, you must change or remove the version setting prior to doing a platform update. This might occur when a security vulnerability is identified for one or more versions of Node.js. When this happens, attemptin g to update to a new version of the platform that doesn't support the configured NodeVersion fails. To avoid needing to create a new environment, change the *NodeVersi on* configuration option to a Node.js version that is supported by both the old platform version and the new one, or remove the option setting, and then perform the platform update.		

GzipCompr ession	Specifies if gzip compression is enabled. If ProxyServer is set to none, then gzip compression is disabled.	`false`	`true`
			
			`false`

Name	Description	Default	Valid values

ProxyServer	Specifies which web server should be used to proxy connections to Node.js. If ProxyServer is set to `none`, then static file mappings doesn't take effect and gzip compression is disabled.	`nginx`	`apache`
			
			`nginx`
			
			`none`

Namespace: `aws:elasticbeanstalk:container:nodejs:staticfiles`

You can use the following namespace to configure the proxy server to serve static files. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application. This reduces the number of requests that your application has to process.

Map a path served by the proxy server to a folder in your source code that contains static assets. Each option that you define in this namespace maps a different path.

Note

Static file settings do not apply if `aws:elasticbeanstalk:container:nodejs::ProxyFiles` is set to none.

Name	Value
Path where the proxy server will serve the files. Example: `/images` to serve files at `*subdomain* .eleasticbeanstalk.com/images .`	Name of the folder containing the files. Example: `staticimages` to serve files from a folder named `staticimages` at the top level of your source bundle.

PHP platform options

Namespace: `aws:elasticbeanstalk:container:php:phpini`

Name	Description	Default	Valid values
document_root	Specify the child directory of your project that is treated as the public-facing web root.	`/`	A blank string is treated as `/`, or specify a string starting with `/`
memory_limit	Amount of memory allocated to the PHP environment.	`256M`	n/a

zlib.output_compression	Specifies whether or not PHP should use compression for output.	`Off`	`On`
			`Off`
			`true`
			`false`

allow_url_fopen	Specifies if PHP's file functions are allowed to retrieve data from remote locations, such as websites or FTP servers.	`On`	`On`
			`Off`
			`true`
			`false`

display_errors	Specifies if error messages should be part of the output.	`Off`	`On` `Off`
max_execution_time	Sets the maximum time, in seconds, a script is allowed to run before it is terminated by the environment.	`60`	`0` to `9223372036854775807` (PHP_INT_MAX)
composer_options	Sets custom options to use when installing dependencies using Composer through the **composer.phar install**	n/a	n/a

Name	Description	Default	Valid values
	command. For more information, see install on the *getcomposer.org* website.		

Namespace: `aws:elasticbeanstalk:environment:proxy`

Name	Description	Default	Valid values
ProxyServer	Set the proxy to use on your environment's instances.	`nginx`	`apache` `nginx`

> **Note**
>
> For more information about the PHP platform, see Using the Elastic Beanstalk PHP platform.

Python platform options

Namespace: `aws:elasticbeanstalk:application:environment`

Name	Description	Default	Valid values
DJANGO_SETTINGS_MODULE	Specifies which settings file to use.	n/a	n/a

See Environment variables and other software settings for more information.

Namespace: `aws:elasticbeanstalk:container:python`

Name	Description	Default	Valid values
WSGIPath	The file that contains the WSGI application. This file must have an `application` callable.	On Amazon Linux 2	n/a

Name	Description	Default	Valid values
		Python platform versions: `application` On Amazon Linux AMI Python platform versions: `application.py`	
NumProcesses	The number of daemon processes that should be started for the process group when running WSGI applications.	1	n/a
NumThreads	The number of threads to be created to handle requests in each daemon process within the process group when running WSGI applications.	15	n/a

Namespace: `aws:elasticbeanstalk:environment:proxy`

Name	Description	Default	Valid values

ProxyServer	Set the proxy to use on your environment's instances.	`nginx`	`apache`
			
			`nginx`

Amazon Linux AMI (pre-Amazon Linux 2) platform options

Namespace: `aws:elasticbeanstalk:container:python:staticfiles`

You can use the following namespace to configure the proxy server to serve static files. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application. This reduces the number of requests that your application has to process.

Map a path served by the proxy server to a folder in your source code that contains static assets. Each option that you define in this namespace maps a different path.

By default, the proxy server in a Python environment serves any files in a folder named `static` at the `/static` path.

Namespace: `aws:elasticbeanstalk:container:python:staticfiles`

Name	Value
Path where the proxy server will serve the files. Example: `/images` to serve files at `*subdomain* .eleasticbeanstalk.com/images .`	Name of the folder containing the files. Example: `staticimages` to serve files from a folder named `staticimages` at the top level of your source bundle.

Ruby platform options

Namespace: `aws:elasticbeanstalk:application:environment`

Name	Description	Default	Valid values

RAILS_SKIP_MIGRATIONS	Specifies whether to run `` `rake db:migrate` `` on behalf of the users' applications; or whether it should be skipped. This is only applicable to Rails 3 applications.	`false`	`true`
			
			`false`

Name	Description	Default	Valid values
RAILS_SKIP_ASSET_COMPILATION	Specifies whether the container should run `` `rake assets:precompile` `` on behalf of the users' applications; or whether it should be skipped. This is also only applicable to Rails 3 applications.	`false`	`true` `false`
BUNDLE_WITHOUT	A colon (`:`) separated list of groups to ignore when installing dependencies from a Gemfile.	`test:development`	n/a
RACK_ENV	Specifies what environment stage an application can be run in. Examples of common environments include development, production, test.	`production`	n/a

See Environment variables and other software settings for more information.

Custom options

Use the `aws:elasticbeanstalk:customoption` namespace to define options and values that can be read in `Resources` blocks in other configuration files. Use custom options to collect user specified settings in a single configuration file.

For example, you may have a complex configuration file that defines a resource that can be configured by the user launching the environment. If you use `Fn::GetOptionSetting` to retrieve the value of a custom option, you can put the definition of that option in a different configuration file, where it is more easily discovered and modified by the user.

Also, because they are configuration options, custom options can be set at the API level to override values set in a configuration file. See Precedence for more information.

Custom options are defined like any other option:

```
option_settings:
  aws:elasticbeanstalk:customoption:
    option name: option value
```

For example, the following configuration file creates an option named `ELBAlarmEmail` and sets the value to `someone@example.com`:

```
option_settings:
  aws:elasticbeanstalk:customoption:
    ELBAlarmEmail: someone@example.com
```

Elsewhere, a configuration file defines an SNS topic that reads the option with `Fn::GetOptionSetting` to populate the value of the `Endpoint` attribute:

```
Resources:
  MySNSTopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Endpoint:
            Fn::GetOptionSetting:
              OptionName: ELBAlarmEmail
              DefaultValue: nobody@example.com
          Protocol: email
```

You can find more example snippets using `Fn::GetOptionSetting` at Adding and customizing Elastic Beanstalk environment resources.

Advanced environment customization with configuration files (`.ebextensions`)

You can add AWS Elastic Beanstalk configuration files (`.ebextensions`) to your web application's source code to configure your environment and customize the AWS resources that it contains. Configuration files are YAML- or JSON-formatted documents with a `.config` file extension that you place in a folder named `.ebextensions` and deploy in your application source bundle.

Example .ebextensions/network-load-balancer.config

This example makes a simple configuration change. It modifies a configuration option to set the type of your environment's load balancer to Network Load Balancer.

```
option_settings:
  aws:elasticbeanstalk:environment:
    LoadBalancerType: network
```

We recommend using YAML for your configuration files, because it's more readable than JSON. YAML supports comments, multi-line commands, several alternatives for using quotes, and more. However, you can make any configuration change in Elastic Beanstalk configuration files identically using either YAML or JSON.

Tip

When you are developing or testing new configuration files, launch a clean environment running the default application and deploy to that. Poorly formatted configuration files will cause a new environment launch to fail unrecoverably.

The `option_settings` section of a configuration file defines values for configuration options. Configuration options let you configure your Elastic Beanstalk environment, the AWS resources in it, and the software that runs your application. Configuration files are only one of several ways to set configuration options.

The `Resources` section lets you further customize the resources in your application's environment, and define additional AWS resources beyond the functionality provided by configuration options. You can add and configure any resources supported by AWS CloudFormation, which Elastic Beanstalk uses to create environments.

The other sections of a configuration file (`packages`, `sources`, `files`, `users`, `groups`, `commands`, `container_commands`, and `services`) let you configure the EC2 instances that are launched in your environment. Whenever a server is launched in your environment, Elastic Beanstalk runs the operations defined in these sections to prepare the operating system and storage system for your application.

For examples of commonly used .ebextensions, see the Elastic Beanstalk Configuration Files Repository.

Requirements

- **Location** – Elastic Beanstalk will process all `.ebextensions` folders present in your deployment. However, we recommend that you place all of your configuration files in a single folder, named `.ebextensions`, in the root of your source bundle. Folders starting with a dot can be hidden by file browsers, so make sure that the folder is added when you create your source bundle. For more information, see Create an Elastic Beanstalk application source bundle.

- **Naming** – Configuration files must have the `.config` file extension.
- **Formatting** – Configuration files must conform to YAML or JSON specifications.

  When using YAML, always use spaces to indent keys at different nesting levels. For more information about YAML, see YAML Ain't Markup Language (YAML™) Version 1.1.
- **Uniqueness** – Use each key only once in each configuration file.

> **Warning**
>
> If you use a key (for example, `option_settings`) twice in the same configuration file, one of the sections will be dropped. Combine duplicate sections into a single section, or place them in separate configuration files.

The process for deploying varies slightly depending on the client that you use to manage your environments. See the following sections for details:

- Elastic Beanstalk console
- EB CLI
- AWS CLI

Topics

Option settings

You can use the `option_settings` key to modify the Elastic Beanstalk configuration and define variables that can be retrieved from your application using environment variables. Some namespaces allow you to extend the number of parameters, and specify the parameter names. For a list of namespaces and configuration options, see Configuration options.

Option settings can also be applied directly to an environment during environment creation or an environment update. Settings applied directly to the environment override the settings for the same options in configuration files. If you remove settings from an environment's configuration, settings in configuration files will take effect. See Precedence for details.

Syntax

The standard syntax for option settings is an array of objects, each having a `namespace`, `option_name` and `value` key.

```
option_settings:
  - namespace:  namespace
    option_name:  option name
    value:  option value
  - namespace:  namespace
    option_name:  option name
    value:  option value
```

The `namespace` key is optional. If you do not specify a namespace, the default used is `aws:elasticbeanstalk:application:environment`:

```
option_settings:
  - option_name:  option name
    value:  option value
  - option_name:  option name
    value:  option value
```

Elastic Beanstalk also supports a shorthand syntax for option settings that lets you specify options as key-value pairs underneath the namespace:

```
option_settings:
  namespace:
    option name: option value
    option name: option value
```

Examples

The following examples set a Tomcat platform-specific option in the `aws:elasticbeanstalk:container:tomcat:jvmoptions` namespace and an environment property named `MYPARAMETER`.

In standard YAML format:

Example .ebextensions/options.config

```
option_settings:
  - namespace:  aws:elasticbeanstalk:container:tomcat:jvmoptions
    option_name:  Xmx
    value:  256m
  - option_name: MYPARAMETER
    value: parametervalue
```

In shorthand format:

Example .ebextensions/options.config

```
option_settings:
  aws:elasticbeanstalk:container:tomcat:jvmoptions:
    Xmx: 256m
  aws:elasticbeanstalk:application:environment:
    MYPARAMETER: parametervalue
```

In JSON:

Example .ebextensions/options.config

```
{
  "option_settings": [
    {
      "namespace": "aws:elasticbeanstalk:container:tomcat:jvmoptions",
      "option_name": "Xmx",
      "value": "256m"
    },
    {
      "option_name": "MYPARAMETER",
      "value": "parametervalue"
    }
  ]
}
```

Customizing software on Linux servers

This section describes the type of information you can include in a configuration file to customize the software on your EC2 instances running Linux. For general information about customizing and configuring your Elastic Beanstalk environments, see Configuring Elastic Beanstalk environments. For information about customizing software on your EC2 instances running Windows, see Customizing software on Windows servers.

You may want to customize and configure the software that your application depends on. You can add commands to be executed during instance provisioning; define Linux users and groups; and download or directly create files on your environment instances. These files might be either dependencies required by the application—for example, additional packages from the yum repository—or they might be configuration files such as a replacement for a proxy configuration file to override specific settings that are defaulted by Elastic Beanstalk.

Notes

- On Amazon Linux 2 platforms, instead of providing files and commands in .ebextensions configuration files, we highly recommend that you use *Buildfile*. *Procfile*, and *platform hooks* whenever possible to configure and run custom code on your environment instances during instance provisioning. For details about these mechanisms, see the section called "Extending Linux platforms".
- YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

Configuration files support the following keys that affect the Linux server your application runs on.

Keys

- Packages
- Groups
- Users
- Sources
- Files
- Commands

- Services
- Container commands
- Example: Using custom Amazon CloudWatch metrics

Keys are processed in the order that they are listed here.

Watch your environment's events while developing and testing configuration files. Elastic Beanstalk ignores a configuration file that contains validation errors, like an invalid key, and doesn't process any of the other keys in the same file. When this happens, Elastic Beanstalk adds a warning event to the event log.

Packages

You can use the `packages` key to download and install prepackaged applications and components.

Syntax

```
packages:
  name of package manager:
    package name: version
    ...
  name of package manager:
    package name: version
    ...
  ...
```

You can specify multiple packages under each package manager's key.

Supported package formats

Elastic Beanstalk currently supports the following package managers: yum, rubygems, python, and rpm. Packages are processed in the following order: rpm, yum, and then rubygems and python. There is no ordering between rubygems and python. Within each package manager, package installation order isn't guaranteed. Use a package manager supported by your operating system.

Note

Elastic Beanstalk supports two underlying package managers for Python, pip and easy_install. However, in the syntax of the configuration file, you must specify the package manager name as python. When you use a configuration file to specify a Python package manager, Elastic Beanstalk uses Python 2.7. If your application relies on a different version of Python, you can specify the packages to install in a requirements.txt file. For more information, see Specifying dependencies using a requirements file on Elastic Beanstalk.

Specifying versions

Within each package manager, each package is specified as a package name and a list of versions. The version can be a string, a list of versions, or an empty string or list. An empty string or list indicates that you want the latest version. For rpm manager, the version is specified as a path to a file on disk or a URL. Relative paths are not supported.

If you specify a version of a package, Elastic Beanstalk attempts to install that version even if a newer version of the package is already installed on the instance. If a newer version is already installed, the deployment fails. Some package managers support multiple versions, but others may not. Please check the documentation for your package manager for more information. If you do not specify a version and a version of the package is already installed, Elastic Beanstalk does not install a new version—it assumes that you want to keep and use the existing version.

Example snippet

The following snippet specifies a version URL for rpm, requests the latest version from yum, and version 0.10.2 of chef from rubygems.

```
packages:
  yum:
    libmemcached: []
    ruby-devel: []
    gcc: []
  rpm:
    epel: http://download.fedoraproject.org/pub/epel/5/i386/epel-release-5-4.noarch.rpm
  rubygems:
    chef: '0.10.2'
```

Groups

You can use the groups key to create Linux/UNIX groups and to assign group IDs. To create a group, add a new key-value pair that maps a new group name to an optional group ID. The groups key can contain one or more group names. The following table lists the available keys.

Syntax

```
groups:
  name of group: {}
  name of group:
    gid: "group id"
```

Options

`gid`

A group ID number.

If a group ID is specified, and the group already exists by name, the group creation will fail. If another group has the specified group ID, the operating system may reject the group creation.

Example snippet

The following snippet specifies a group named groupOne without assigning a group ID and a group named groupTwo that specified a group ID value of 45.

```
groups:
  groupOne: {}
  groupTwo:
    gid: "45"
```

Users

You can use the `users` key to create Linux/UNIX users on the EC2 instance.

Syntax

```
users:
  name of user:
    groups:
      - name of group
    uid: "id of the user"
    homeDir: "user's home directory"
```

Options

`uid`

A user ID. The creation process fails if the user name exists with a different user ID. If the user ID is already assigned to an existing user, the operating system may reject the creation request.

`groups`

A list of group names. The user is added to each group in the list.

`homeDir`

The user's home directory.

Users are created as noninteractive system users with a shell of `/sbin/nologin`. This is by design and cannot be modified.

Example snippet

```
users:
  myuser:
    groups:
      - group1
      - group2
    uid: "50"
    homeDir: "/tmp"
```

Sources

You can use the `sources` key to download an archive file from a public URL and unpack it in a target directory on the EC2 instance.

Syntax

```
sources:
  target directory: location of archive file
```

Supported formats

Supported formats are tar, tar+gzip, tar+bz2, and zip. You can reference external locations such as Amazon Simple Storage Service (Amazon S3) (e.g., `https://amzn-s3-demo-bucket.s3.amazonaws.com/myobject`) as long as the URL is publicly accessible.

Example snippet

The following example downloads a public .zip file from an Amazon S3 bucket and unpacks it into `/etc/myapp`:

```
sources:
  /etc/myapp: https://amzn-s3-demo-bucket.s3.amazonaws.com/myobject
```

Note

Multiple extractions should not reuse the same target path. Extracting another source to the same target path will replace rather than append to the contents.

Files

You can use the `files` key to create files on the EC2 instance. The content can be either inline in the configuration file, or the content can be pulled from a URL. The files are written to disk in lexicographic order.

You can use the `files` key to download private files from Amazon S3 by providing an instance profile for authorization.

If the file path you specify already exists on the instance, the existing file is retained with the extension `.bak` appended to its name.

Syntax

```
files:
  "target file location on disk":
     mode: "six-digit octal value"
     owner: name of owning user for file
     group: name of owning group for file
     source: URL
     authentication: authentication name:

  "target file location on disk":
     mode: "six-digit octal value"
     owner: name of owning user for file
     group: name of owning group for file
     content: |
      # this is my
      # file content
     encoding: encoding format
     authentication: authentication name:
```

Options

`content`

String content to add to the file. Specify either `content` or `source`, but not both.

`source`

URL of a file to download. Specify either `content` or `source`, but not both.

`encoding`

The encoding format of the string specified with the `content` option.

Valid values: `plain` | `base64`

`group`

Linux group that owns the file.

`owner`

Linux user that owns the file.

`mode`

A six-digit octal value representing the mode for this file. Not supported for Windows systems. Use the first three digits for symlinks and the last three digits for setting permissions. To create a symlink, specify `120xxx`, where `xxx` defines the permissions of the target file. To specify permissions for a file, use the last three digits, such as `000644`.

`authentication`

The name of a AWS CloudFormation authentication method to use. You can add authentication methods to the Auto Scaling group metadata with the Resources key. See below for an example.

Example snippet

```
files:
  "/home/ec2-user/myfile" :
    mode: "000755"
    owner: root
    group: root
    source: http://foo.bar/myfile

  "/home/ec2-user/myfile2" :
    mode: "000755"
    owner: root
    group: root
    content: |
      this is my
      file content
```

Example using a symlink. This creates a link `/tmp/myfile2.txt` that points at the existing file `/tmp/myfile1.txt`.

```
files:
  "/tmp/myfile2.txt" :
    mode: "120400"
    content: "/tmp/myfile1.txt"
```

The following example uses the Resources key to add an authentication method named S3Auth and uses it to download a private file from an Amazon S3 bucket:

```
Resources:
  AWSEBAutoScalingGroup:
    Metadata:
      AWS::CloudFormation::Authentication:
        S3Auth:
          type: "s3"
          buckets: ["amzn-s3-demo-bucket2"]
          roleName:
            "Fn::GetOptionSetting":
              Namespace: "aws:autoscaling:launchconfiguration"
              OptionName: "IamInstanceProfile"
              DefaultValue: "aws-elasticbeanstalk-ec2-role"

files:
  "/tmp/data.json" :
    mode: "000755"
    owner: root
    group: root
    authentication: "S3Auth"
    source: https://elasticbeanstalk-us-west-2-123456789012.s3-us-west-2.amazonaws.com/
data.json
```

Commands

You can use the `commands` key to execute commands on the EC2 instance. The commands run before the application and web server are set up and the application version file is extracted.

The specified commands run as the root user, and are processed in alphabetical order by name. By default, commands run in the root directory. To run commands from another directory, use the `cwd` option.

To troubleshoot issues with your commands, you can find their output in instance logs.

Syntax

```
commands:
  command name:
    command: command to run
    cwd: working directory
    env:
      variable name: variable value
    test: conditions for command
    ignoreErrors: true
```

Options

`command`

Either an array (block sequence collection in YAML syntax) or a string specifying the command to run. Some important notes:

- If you use a string, you don't need to enclose the entire string in quotes. If you do use quotes, escape literal occurrences of the same type of quote.
- If you use an array, you don't need to escape space characters or enclose command parameters in quotes. Each array element is a single command argument. Don't use an array to specify multiple commands.

The following examples are all equivalent:

```
commands:
  command1:
    command: git commit -m "This is a comment."
  command2:
    command: "git commit -m \"This is a comment.\""
  command3:
    command: 'git commit -m "This is a comment."'
  command4:
    command:
      - git
      - commit
      - -m
      - This is a comment.
```

To specify multiple commands, use a literal block scalar, as shown in the following example.

```
commands:
  command block:
    command: |
      git commit -m "This is a comment."
      git push
```

`env`

(Optional) Sets environment variables for the command. This property overwrites, rather than appends, the existing environment.

`cwd`

(Optional) The working directory. If not specified, commands run from the root directory (/).

`test`

(Optional) A command that must return the value `true` (exit code 0) in order for Elastic Beanstalk to process the command, such as a shell script, contained in the `command` key.

`ignoreErrors`

(Optional) A boolean value that determines if other commands should run if the command contained in the `command` key fails (returns a nonzero value). Set this value to `true` if you want to continue running commands even if the command fails. Set it to `false` if you want to stop running commands if the command fails. The default value is `false`.

Example snippet

The following example snippet runs a Python script.

```
commands:
  python_install:
    command: myscript.py
    cwd: /home/ec2-user
    env:
      myvarname: myvarvalue
    test: "[ -x /usr/bin/python ]"
```

Services

You can use the `services` key to define which services should be started or stopped when the instance is launched. The `services` key also allows you to specify dependencies on sources, packages, and files so that if a restart is needed due to files being installed, Elastic Beanstalk takes care of the service restart.

Syntax

```
services:
  sysvinit:
    name of service:
      enabled: "true"
      ensureRunning: "true"
      files:
        - "file name"
      sources:
        - "directory"
      packages:
        name of package manager:
          "package name[: version]"
      commands:
        - "name of command"
```

Options

`ensureRunning`

Set to `true` to ensure that the service is running after Elastic Beanstalk finishes.

Set to `false` to ensure that the service is not running after Elastic Beanstalk finishes.

Omit this key to make no changes to the service state.

`enabled`

Set to `true` to ensure that the service is started automatically upon boot.

Set to `false` to ensure that the service is not started automatically upon boot.

Omit this key to make no changes to this property.

`files`

A list of files. If Elastic Beanstalk changes one directly via the files block, the service is restarted.

`sources`

A list of directories. If Elastic Beanstalk expands an archive into one of these directories, the service is restarted.

`packages`

A map of the package manager to a list of package names. If Elastic Beanstalk installs or updates one of these packages, the service is restarted.

`commands`

A list of command names. If Elastic Beanstalk runs the specified command, the service is restarted.

Example snippet

The following is an example snippet:

```
services:
  sysvinit:
    myservice:
      enabled: true
      ensureRunning: true
```

Container commands

You can use the `container_commands` key to execute commands that affect your application source code. Container commands run after the application and web server have been set up and the application version archive has been extracted, but before the application version is deployed. Non-container commands and other customization operations are performed prior to the application source code being extracted.

The specified commands run as the root user, and are processed in alphabetical order by name. Container commands are run from the staging directory, where your source code is extracted prior to being deployed to the application server. Any changes you make to your source code in the staging directory with a container command will be included when the source is deployed to its final location.

Note

The output of your container commands are logged in the `cfn-init-cmd.log` instance log. For more information about retrieving and viewing instance logs, see Viewing logs from Amazon EC2 instances.

You can use `leader_only` to only run the command on a single instance, or configure a `test` to only run the command when a test command evaluates to `true`. Leader-only container commands are only executed during environment creation and deployments, while other commands and server customization operations are performed every time an instance is provisioned or updated. Leader-only container commands are not executed due to launch configuration changes, such as a change in the AMI Id or instance type.

Syntax

```
container_commands:
  name of container_command:
    command: "command to run"
    leader_only: true
  name of container_command:
    command: "command to run"
```

Options

`command`

A string or array of strings to run.

`env`

(Optional) Set environment variables prior to running the command, overriding any existing value.

`cwd`

(Optional) The working directory. By default, this is the staging directory of the unzipped application.

`leader_only`

(Optional) Only run the command on a single instance chosen by Elastic Beanstalk. Leader-only container commands are run before other container commands. A command can be leader-only or have a `test`, but not both (`leader_only` takes precedence).

`test`

(Optional) Run a test command that must return the `true` in order to run the container command. A command can be leader-only or have a `test`, but not both (`leader_only` takes precedence).

`ignoreErrors`

(Optional) Do not fail deployments if the container command returns a value other than 0 (success). Set to `true` to enable.

Example snippet

The following is an example snippet.

```
container_commands:
  collectstatic:
    command: "django-admin.py collectstatic --noinput"
  01syncdb:
    command: "django-admin.py syncdb --noinput"
    leader_only: true
  02migrate:
    command: "django-admin.py migrate"
    leader_only: true
  99customize:
    command: "scripts/customize.sh"
```

Example: Using custom Amazon CloudWatch metrics

This topic provides a configuration example that integrates Elastic Beanstalk metrics with Amazon CloudWatch agent for platforms based on Amazon Linux 2 and later. The configuration example uses files and commands in an `.ebextensions` configuration file.

Amazon CloudWatch is a web service that enables you to monitor, manage, and publish various metrics, as well as configure alarm actions based on data from metrics. You can define custom metrics for your own use, and Elastic Beanstalk will push those metrics to Amazon CloudWatch. Once Amazon CloudWatch contains your custom metrics, you can view those in the Amazon CloudWatch console.

The Amazon CloudWatch agent

The Amazon CloudWatch agent enables CloudWatch metric and log collection from both Amazon EC2 instances and on-premises servers across operating systems. The agent supports metrics collected at the system level. It also supports custom log and metric collection from your applications or services. For more information about the Amazon CloudWatch agent, see Collecting metrics and logs with the CloudWatch agent in the *Amazon CloudWatch User Guide*.

Note

Elastic Beanstalk Enhanced Health Reporting has native support for publishing a wide range of instance and environment metrics to CloudWatch. See Publishing Amazon CloudWatch custom metrics for an environment for details.

Topics

.Ebextensions configuration file

This example uses files and commands in an .ebextensions configuration file to configure and run the Amazon CloudWatch agent on the Amazon Linux 2 platform. The agent is prepackaged with Amazon Linux 2. If you're using a different operating system, additional steps for installing the agent may be necessary. For more information, see Installing the CloudWatch agent in the *Amazon CloudWatch User Guide*.

To use this sample, save it to a file named `cloudwatch.config` in a directory named `.ebextensions` at the top level of your project directory, then deploy your application using the Elastic Beanstalk console (include the .ebextensions directory in your source bundle) or the EB CLI.

For more information about configuration files, see Advanced environment customization with configuration files (`.ebextensions`).

This file has two sections:

- `files` — This section adds the agent configuration file. It indicates which metrics and logs the agent should send to Amazon CloudWatch. In this example, we're only sending the *mem_used_percent* metric. For a complete listing of system level metrics supported by the Amazon CloudWatch agent, see Metrics collected by the CloudWatch agent in the *Amazon CloudWatch User Guide*.
- `container_commands` — This section contains the command that starts the agent, passing in the configuration file as a parameter. For more details about `container_commands`, see Container commands.

.ebextensions/cloudwatch.config

```
files:
  "/opt/aws/amazon-cloudwatch-agent/bin/config.json":
    mode: "000600"
    owner: root
    group: root
    content: |
      {
        "agent": {
          "metrics_collection_interval": 60,
          "run_as_user": "root"
        },
        "metrics": {
          "namespace": "System/Linux",
          "append_dimensions": {
            "AutoScalingGroupName": "${aws:AutoScalingGroupName}"
          },
          "metrics_collected": {
            "mem": {
              "measurement": [
                "mem_used_percent"
              ]
            }
          }
        }
      }
```

```
container_commands:
  start_cloudwatch_agent:
    command: /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a
 append-config -m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/bin/config.json
```

Permissions

The instances in your environment need the proper IAM permissions in order to publish custom Amazon CloudWatch metrics using the Amazon CloudWatch agent. You grant permissions to your environment's instances by adding them to the environment's instance profile. You can add permissions to the instance profile before or after deploying your application.

To grant permissions to publish CloudWatch metrics

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Roles**.
3. Choose your environment's instance profile role. By default, when you create an environment with the Elastic Beanstalk console or EB CLI, this is `aws-elasticbeanstalk-ec2-role`.
4. Choose the **Permissions** tab.
5. Under **Permissions Policies**, in the **Permissions** section, choose **Attach policies**.
6. Under **Attach Permissions**, choose the AWS managed policy **CloudWatchAgentServerPolicy**. Then click **Attach policy**.

For more information about managing policies, see Working with Policies in the *IAM User Guide*.

Viewing metrics in the CloudWatch console

After deploying the CloudWatch configuration file to your environment, check the Amazon CloudWatch console to view your metrics. Custom metrics will be located in the **CWAgent** namespace.

For more information, see Viewing available metrics in the *Amazon CloudWatch User Guide*.

Customizing software on Windows servers

You may want to customize and configure the software that your application depends on. These files could be either dependencies required by the application—for example, additional packages or services that need to be run. For general information on customizing and configuring your Elastic Beanstalk environments, see Configuring Elastic Beanstalk environments.

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

Configuration files support the following keys that affect the Windows server on which your application runs.

Keys

- Packages
- Sources
- Files
- Commands
- Services
- Container commands

Keys are processed in the order that they are listed here.

Note

Older (non-versioned) .NET platform versions do not process configuration files in the correct order. Learn more at Migrating across major versions of the Elastic Beanstalk Windows server platform.

Watch your environment's events while developing and testing configuration files. Elastic Beanstalk ignores a configuration file that contains validation errors, like an invalid key, and doesn't process any of the other keys in the same file. When this happens, Elastic Beanstalk adds a warning event to the event log.

Packages

Use the `packages` key to download and install prepackaged applications and components.

In Windows environments, Elastic Beanstalk supports downloading and installing MSI packages. (Linux environments support additional package managers. For details, see Packages on the *Customizing Software on Linux Servers* page.)

You can reference any external location, such as an Amazon Simple Storage Service (Amazon S3) object, as long as the URL is publicly accessible.

If you specify several `msi:` packages, their installation order isn't guaranteed.

Syntax

Specify a name of your choice as the package name, and a URL to an MSI file location as the value. You can specify multiple packages under the `msi:` key.

```
packages:
  msi:
    package name: package url
    ...
```

Examples

The following example specifies a URL to download **mysql** from `https://dev.mysql.com/`.

```
packages:
  msi:
    mysql: https://dev.mysql.com/get/Downloads/Connector-Net/mysql-connector-
net-8.0.11.msi
```

The following example specifies an Amazon S3 object as the MSI file location.

```
packages:
  msi:
    mymsi: https://amzn-s3-demo-bucket.s3.amazonaws.com/myobject.msi
```

Sources

Use the `sources` key to download an archive file from a public URL and unpack it in a target directory on the EC2 instance.

Syntax

```
sources:
  target directory: location of archive file
```

Supported formats

In Windows environments, Elastic Beanstalk supports the .zip format. (Linux environments support additional formats. For details, see Sources on the *Customizing Software on Linux Servers* page.)

You can reference any external location, such as an Amazon Simple Storage Service (Amazon S3) object, as long as the URL is publicly accessible.

Example

The following example downloads a public .zip file from an Amazon S3 and unpacks it into `c:/myproject/myapp`.

```
sources:
  "c:/myproject/myapp": https://amzn-s3-demo-bucket.s3.amazonaws.com/myobject.zip
```

Files

Use the `files` key to create files on the EC2 instance. The content can be either inline in the configuration file, or from a URL. The files are written to disk in lexicographic order. To download private files from Amazon S3, provide an instance profile for authorization.

Syntax

```
files:
  "target file location on disk":
    source: URL
    authentication: authentication name:

  "target file location on disk":
    content: |
      this is my content
    encoding: encoding format
```

Options

`content`

(Optional) A string.

`source`

(Optional) The URL from which the file is loaded. This option cannot be specified with the content key.

`encoding`

(Optional) The encoding format. This option is only used for a provided content key value. The default value is `plain`.

Valid values: `plain` | `base64`

`authentication`

(Optional) The name of a AWS CloudFormation authentication method to use. You can add authentication methods to the Auto Scaling group metadata with the Resources key.

Examples

The following example shows the two ways to provide file content: from a URL, or inline in the configuration file.

```
files:
  "c:\\targetdirectory\\targetfile.txt":
    source: http://foo.bar/myfile

  "c:/targetdirectory/targetfile.txt":
    content: |
      # this is my file
      # with content
```

Note

If you use a backslash (\) in your file path, you must precede that with another backslash (the escape character) as shown in the previous example.

The following example uses the Resources key to add an authentication method named S3Auth and uses it to download a private file from an Amazon S3 :

```
files:
  "c:\\targetdirectory\\targetfile.zip":
    source: https://elasticbeanstalk-us-east-2-123456789012.s3.amazonaws.com/prefix/
myfile.zip
    authentication: S3Auth

Resources:
  AWSEBAutoScalingGroup:
    Metadata:
      AWS::CloudFormation::Authentication:
        S3Auth:
          type: "s3"
          s: ["amzn-s3-demo-bucket"]
          roleName:
            "Fn::GetOptionSetting":
              Namespace: "aws:autoscaling:launchconfiguration"
              OptionName: "IamInstanceProfile"
              DefaultValue: "aws-elasticbeanstalk-ec2-role"
```

Commands

Use the `commands` key to execute commands on the EC2 instance. The commands are processed in alphabetical order by name, and they run before the application and web server are set up and the application version file is extracted.

The specified commands run as the Administrator user.

To troubleshoot issues with your commands, you can find their output in instance logs.

Syntax

```
commands:
  command name:
    command: command to run
```

Options

`command`

Either an array or a string specifying the command to run. If you use an array, you don't need to escape space characters or enclose command parameters in quotation marks.

`cwd`

(Optional) The working directory. By default, Elastic Beanstalk attempts to find the directory location of your project. If not found, it uses `c:\Windows\System32` as the default.

`env`

(Optional) Sets environment variables for the command. This property overwrites, rather than appends, the existing environment.

`ignoreErrors`

(Optional) A Boolean value that determines if other commands should run if the command contained in the `command` key fails (returns a nonzero value). Set this value to `true` if you want to continue running commands even if the command fails. Set it to `false` if you want to stop running commands if the command fails. The default value is `false`.

`test`

(Optional) A command that must return the value `true` (exit code 0) in order for Elastic Beanstalk to process the command contained in the `command` key.

`waitAfterCompletion`

(Optional) Seconds to wait after the command completes before running the next command. The default value is **`60`** seconds. You can also specify **`forever`**.

Important

System reboots during deployment are not supported. If the system reboots as a result of a command, instance initialization will fail, causing the deployment to fail. As a workaround, you can use this `.ebextensions configuration` to schedule reboots after deployment is complete.

Example

The following example saves the output of the `set` command to the specified file. If there is a subsequent command, Elastic Beanstalk runs that command immediately after this command completes. If this command requires a reboot, Elastic Beanstalk reboots the instance immediately after the command completes.

```
commands:
  test:
    command: set > c:\\myapp\\set.txt
    waitAfterCompletion: 0
```

Services

Use the `services` key to define which services should be started or stopped when the instance is launched. The `services` key also enables you to specify dependencies on sources, packages, and files so that if a restart is needed due to files being installed, Elastic Beanstalk takes care of the service restart.

Syntax

```
services:
  windows:
    name of service:
      files:
        - "file name"
      sources:
        - "directory"
      packages:
        name of package manager:
            "package name[: version]"
      commands:
        - "name of command"
```

Options

`ensureRunning`

(Optional) Set to `true` to ensure that the service is running after Elastic Beanstalk finishes.

Set to `false` to ensure that the service is not running after Elastic Beanstalk finishes.

Omit this key to make no changes to the service state.

`enabled`

(Optional) Set to `true` to ensure that the service is started automatically upon boot.

Set to `false` to ensure that the service is not started automatically upon boot.

Omit this key to make no changes to this property.

`files`

A list of files. If Elastic Beanstalk changes one directly via the files block, the service is restarted.

`sources`

A list of directories. If Elastic Beanstalk expands an archive into one of these directories, the service is restarted.

`packages`

A map of the package manager to a list of package names. If Elastic Beanstalk installs or updates one of these packages, the service is restarted.

`commands`

A list of command names. If Elastic Beanstalk runs the specified command, the service is restarted.

Example

```
services: 
  windows:
    myservice:
      enabled: true
      ensureRunning: true
```

Container commands

Use the `container_commands` key to execute commands that affect your application source code. Container commands run after the application and web server have been set up and the application version archive has been extracted, but before the application version is deployed. Non-container commands and other customization operations are performed prior to the application source code being extracted.

Container commands are run from the staging directory, where your source code is extracted prior to being deployed to the application server. Any changes you make to your source code in the staging directory with a container command will be included when the source is deployed to its final location.

To troubleshoot issues with your container commands, you can find their output in instance logs.

Use the `leader_only` option to only run the command on a single instance, or configure a `test` to only run the command when a test command evaluates to `true`. Leader-only container commands are only executed during environment creation and deployments, while other commands and server customization operations are performed every time an instance is provisioned or updated. Leader-only container commands are not executed due to launch configuration changes, such as a change in the AMI Id or instance type.

Syntax

```
container_commands:
  name of container_command:
    command: command to run
```

Options

`command`

A string or array of strings to run.

`env`

(Optional) Set environment variables prior to running the command, overriding any existing value.

`cwd`

(Optional) The working directory. By default, this is the staging directory of the unzipped application.

`leader_only`

(Optional) Only run the command on a single instance chosen by Elastic Beanstalk. Leader-only container commands are run before other container commands. A command can be leader-only or have a `test`, but not both (`leader_only` takes precedence).

`test`

(Optional) Run a test command that must return the `true` in order to run the container command. A command can be leader-only or have a `test`, but not both (`leader_only` takes precedence).

`ignoreErrors`

(Optional) Do not fail deployments if the container command returns a value other than 0 (success). Set to `true` to enable.

`waitAfterCompletion`

(Optional) Seconds to wait after the command completes before running the next command. The default value is **`60`** seconds. You can also specify **`forever`**.

Important

System reboots during deployment are not supported. If the system reboots as a result of a command, instance initialization will fail, causing the deployment to fail.

As a workaround, you can use this `.ebextensions configuration` to schedule reboots after deployment is complete.

Example

The following example saves the output of the `set` command to the specified file. Elastic Beanstalk runs the command on one instance, and reboots the instance immediately after the command completes.

```
container_commands:
  foo:
    command: set > c:\\myapp\\set.txt
    leader_only: true
    waitAfterCompletion: 0
```

Adding and customizing Elastic Beanstalk environment resources

You may want to customize your environment resources that are part of your Elastic Beanstalk environment. For example, you may want to add an Amazon SQS queue and an alarm on queue depth, or you might want to add an Amazon ElastiCache cluster. You can easily customize your environment at the same time that you deploy your application version by including a configuration file with your source bundle.

You can use the `Resources` key in a configuration file to create and customize AWS resources in your environment. Resources defined in configuration files are added to the AWS CloudFormation template used to launch your environment. All AWS CloudFormation resources types are supported.

> **Note**
>
> Whenever you add a resource that isn't managed by Elastic Beanstalk, be sure to add a user policy with the appropriate permissions to your AWS Identity and Access Management (IAM) users. The managed user policies that Elastic Beanstalk provides only cover permissions to Elastic Beanstalk-managed resources.

For example, the following configuration file adds an Auto Scaling lifecycle hook to the default Auto Scaling group created by Elastic Beanstalk:

~/my-app/.ebextensions/as-hook.config

```
Resources:
  hookrole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument: {
               "Version" : "2012-10-17",
               "Statement": [ {
                  "Effect": "Allow",
                  "Principal": {
                     "Service": [ "autoscaling.amazonaws.com" ]
                  },
                  "Action": [ "sts:AssumeRole" ]
               } ]
            }
      Policies: [ {
               "PolicyName": "SNS",
               "PolicyDocument": {
                      "Version": "2012-10-17",
                      "Statement": [{
                          "Effect": "Allow",
                          "Resource": "*",
                          "Action": [
                              "sqs:SendMessage",
                              "sqs:GetQueueUrl",
                              "sns:Publish"
                          ]
```

```
                        }
                      ]
                  }
               } ]
  hooktopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Endpoint: "my-email@example.com"
          Protocol: email
  lifecyclehook:
    Type: AWS::AutoScaling::LifecycleHook
    Properties:
      AutoScalingGroupName: { "Ref" : "AWSEBAutoScalingGroup" }
      LifecycleTransition: autoscaling:EC2_INSTANCE_TERMINATING
      NotificationTargetARN: { "Ref" : "hooktopic" }
      RoleARN: { "Fn::GetAtt" : [ "hookrole", "Arn"] }
```

This example defines three resources, `hookrole`, `hooktopic` and `lifecyclehook`. The first two resources are an IAM role, which grants Amazon EC2 Auto Scaling permission to publish messages to Amazon SNS, and an SNS topic, which relays messages from the Auto Scaling group to an email address. Elastic Beanstalk creates these resources with the specified properties and types.

The final resource, `lifecyclehook`, is the lifecycle hook itself:

```
  lifecyclehook:
    Type: AWS::AutoScaling::LifecycleHook
    Properties:
      AutoScalingGroupName: { "Ref" : "AWSEBAutoScalingGroup" }
      LifecycleTransition: autoscaling:EC2_INSTANCE_TERMINATING
      NotificationTargetARN: { "Ref" : "hooktopic" }
      RoleARN: { "Fn::GetAtt" : [ "hookrole", "Arn"] }
```

The lifecycle hook definition uses two functions to populate values for the hook's properties. `{ "Ref" : "AWSEBAutoScalingGroup" }` retrieves the name of the Auto Scaling group created by Elastic Beanstalk for the environment. `AWSEBAutoScalingGroup` is one of the standard resource names provided by Elastic Beanstalk.

For `AWS::IAM::Role`, `Ref` only returns the name of the role, not the ARN. To get the ARN for the `RoleARN` parameter, you use another intrinsic function, `Fn::GetAtt` instead, which can get any attribute from a resource. `RoleARN: { "Fn::GetAtt" : [ "hookrole", "Arn"] }` gets the `Arn` attribute from the `hookrole` resource.

`{ "Ref" : "hooktopic" }` gets the ARN of the Amazon SNS topic created earlier in the configuration file. The value returned by `Ref` varies per resource type and can be found in the AWS CloudFormation User Guide topic for the AWS::SNS::Topic resource type.

Modifying the resources that Elastic Beanstalk creates for your environment

The resources that Elastic Beanstalk creates for your environment have names. You can use these names to get information about the resources with a function, or modify properties on the resources to customize their behavior. This topic describes the AWS resources that Elastic Beanstalk uses in the different types of environments.

> **Note**
>
> The previous topic Custom resources provides some uses cases and examples for customizing environment resources. You can also find more examples of configuration files in the later topic Custom resource examples.

Web server environments have the following resources.

Web server environments

- `AWSEBAutoScalingGroup` (AWS::AutoScaling::AutoScalingGroup) – The Auto Scaling group attached to your environment.
- One of the following two resources.
  - `AWSEBAutoScalingLaunchConfiguration` (AWS::AutoScaling::LaunchConfiguration) – The launch configuration attached to your environment's Auto Scaling group.
  - `AWSEBEC2LaunchTemplate` (AWS::EC2::LaunchTemplate) – The Amazon EC2 launch template used by your environment's Auto Scaling group.

  > **Note**
  >
  > If your environment uses functionality that requires Amazon EC2 launch templates, and your user policy lacks the required permissions, creating or updating the environment

might fail. Use the **AdministratorAccess-AWSElasticBeanstalk** managed user policy, or add the required permissions to your custom policy.

- `AWSEBEnvironmentName` (AWS::ElasticBeanstalk::Environment) – Your environment.
- `AWSEBSecurityGroup` (AWS::EC2::SecurityGroup) – The security group attached to your Auto Scaling group.
- `AWSEBRDSDatabase` (AWS::RDS::DBInstance) – The Amazon RDS DB instance attached to your environment (if applicable).

In a load-balanced environment, you can access additional resources related to the load balancer. Classic load balancers have a resource for the load balancer and one for the security group attached to it. Application and network load balancers have additional resources for the load balancer's default listener, listener rule, and target group.

Load-balanced environments

- `AWSEBLoadBalancer` (AWS::ElasticLoadBalancing::LoadBalancer) – Your environment's classic load balancer.
- `AWSEBV2LoadBalancer` (AWS::ElasticLoadBalancingV2::LoadBalancer) – Your environment's application or network load balancer.
- `AWSEBLoadBalancerSecurityGroup` (AWS::EC2::SecurityGroup) – In a custom Amazon Virtual Private Cloud (Amazon VPC) only, the name of the security group that Elastic Beanstalk creates for the load balancer. In a default VPC or EC2 classic, Elastic Load Balancing assigns a default security group to the load balancer.
- `AWSEBV2LoadBalancerListener` (AWS::ElasticLoadBalancingV2::Listener) – A listener that allows the load balancer to check for connection requests and forward them to one or more target groups.
- `AWSEBV2LoadBalancerListenerRule` (AWS::ElasticLoadBalancingV2::ListenerRule) – Defines which requests an Elastic Load Balancing listener takes action on and the action that it takes.
- `AWSEBV2LoadBalancerTargetGroup` (AWS::ElasticLoadBalancingV2::TargetGroup) – An Elastic Load Balancing target group that routes requests to one or more registered targets, such as Amazon EC2 instances.

Worker environments have resources for the SQS queue that buffers incoming requests, and a Amazon DynamoDB table that the instances use for leader election.

Worker environments

- `AWSEBWorkerQueue` (AWS::SQS::Queue) – The Amazon SQS queue from which the daemon pulls requests that need to be processed.
- `AWSEBWorkerDeadLetterQueue` (AWS::SQS::Queue) – The Amazon SQS queue that stores messages that cannot be delivered or otherwise were not successfully processed by the daemon.
- `AWSEBWorkerCronLeaderRegistry` (AWS::DynamoDB::Table) – The Amazon DynamoDB table that is the internal registry used by the daemon for periodic tasks.

Other AWS CloudFormation template keys

We've already introduced configuration file keys from AWS CloudFormation such as `Resources`, `files`, and `packages`. Elastic Beanstalk adds the contents of configurations files to the AWS CloudFormation template that supports your environment, so you can use other AWS CloudFormation sections to perform advanced tasks in your configuration files.

Keys

- Parameters
- Outputs
- Mappings

Parameters

Parameters are an alternative to Elastic Beanstalk's own custom options that you can use to define values that you use in other places in your configuration files. Like custom options, you can use parameters to gather user configurable values in one place. Unlike custom options, you can not use Elastic Beanstalk's API to set parameter values, and the number of parameters you can define in a template is limited by AWS CloudFormation.

One reason you might want to use parameters is to make your configuration files double as AWS CloudFormation templates. If you use parameters instead of custom options, you can use the configuration file to create the same resource in AWS CloudFormation as its own stack. For example, you could have a configuration file that adds an Amazon EFS file system to your environment for testing, and then use the same file to create an independent file system that isn't tied to your environment's lifecycle for production use.

The following example shows the use of parameters to gather user-configurable values at the top of a configuration file.

Example Loadbalancer-accesslogs-existingbucket.config – Parameters

```
Parameters:
  bucket:
    Type: String
    Description: "Name of the Amazon S3 bucket in which to store load balancer logs"
    Default: "amzn-s3-demo-bucket"
  bucketprefix:
    Type: String
    Description: "Optional prefix. Can't start or end with a /, or contain the word
 AWSLogs"
    Default: ""
```

Outputs

You can use an `Outputs` block to export information about created resources to AWS CloudFormation. You can then use the `Fn::ImportValue` function to pull the value into a AWS CloudFormation template outside of Elastic Beanstalk.

The following example creates an Amazon SNS topic and exports its ARN to AWS CloudFormation with the name `NotificationTopicArn`.

Example sns-topic.config

```
Resources:
  NotificationTopic:
    Type: AWS::SNS::Topic

Outputs:
  NotificationTopicArn:
    Description: Notification topic ARN
    Value: { "Ref" : "NotificationTopic" }
    Export:
      Name: NotificationTopicArn
```

In a configuration file for a different environment, or a AWS CloudFormation template outside of Elastic Beanstalk, you can use the `Fn::ImportValue` function to get the exported ARN. This example assigns the exported value to an environment property named `TOPIC_ARN`.

Example env.config

```
option_settings:
  aws:elasticbeanstalk:application:environment:
    TOPIC_ARN: '`{ "Fn::ImportValue" : "NotificationTopicArn" }`'
```

Mappings

You can use a mapping to store key-value pairs organized by namespace. A mapping can help you organize values that you use throughout your configs, or change a parameter value depending on another value. For example, the following configuration sets the value of an account ID parameter based on the current region.

Example Loadbalancer-accesslogs-newbucket.config – Mappings

```
Mappings:
  Region2ELBAccountId:
    us-east-1:
      AccountId: "111122223333"
    us-west-2:
      AccountId: "444455556666"
    us-west-1:
      AccountId: "123456789012"
    eu-west-1:
      AccountId: "777788889999"
...
            Principal:
              AWS:
                ? "Fn::FindInMap"
                :
                  - Region2ELBAccountId
                  -
                    Ref: "AWS::Region"
                  - AccountId
```

Functions

You can use functions in your configuration files to populate values for resource properties with information from other resources or from Elastic Beanstalk configuration option settings. Elastic Beanstalk supports AWS CloudFormation functions (`Ref`, `Fn::GetAtt`, `Fn::Join`), and one Elastic Beanstalk-specific function, `Fn::GetOptionSetting`.

Functions

- Ref
- Fn::GetAtt
- Fn::Join
- Fn::GetOptionSetting

Ref

Use `Ref` to retrieve the default string representation of an AWS resource. The value returned by `Ref` depends on the resource type, and sometimes depends on other factors as well. For example, a security group (AWS::EC2::SecurityGroup) returns either the name or ID of the security group, depending on if the security group is in a default Amazon Virtual Private Cloud (Amazon VPC), EC2 classic, or a custom VPC.

```
{ "Ref" : "resource name" }
```

Note

For details on each resource type, including the return value(s) of `Ref`, see AWS Resource Types Reference in the *AWS CloudFormation User Guide*.

From the sample Auto Scaling lifecycle hook:

```
Resources:
  lifecyclehook:
    Type: AWS::AutoScaling::LifecycleHook
    Properties:
      AutoScalingGroupName: { "Ref" : "AWSEBAutoScalingGroup" }
```

You can also use `Ref` to retrieve the value of a AWS CloudFormation parameter defined elsewhere in the same file or in a different configuration file.

Fn::GetAtt

Use `Fn::GetAtt` to retrieve the value of an attribute on an AWS resource.

```
{ "Fn::GetAtt" : [ "resource name", "attribute name"] }
```

From the sample Auto Scaling lifecycle hook:

```
Resources:
  lifecyclehook:
    Type: AWS::AutoScaling::LifecycleHook
    Properties:
      RoleARN: { "Fn::GetAtt" : [ "hookrole", "Arn"] }
```

See Fn::GetAtt for more information.

Fn::Join

Use `Fn::Join` to combine strings with a delimiter. The strings can be hard-coded or use the output from `Fn::GetAtt` or `Ref`.

```
{ "Fn::Join" : [ "delimiter", [ "string1", "string2" ] ] }
```

See Fn::Join for more information.

Fn::GetOptionSetting

Use `Fn::GetOptionSetting` to retrieve the value of a configuration option setting applied to the environment.

```
"Fn::GetOptionSetting":
  Namespace: "namespace"
  OptionName: "option name"
  DefaultValue: "default value"
```

From the storing private keys example:

```
Resources:
  AWSEBAutoScalingGroup:
    Metadata:
      AWS::CloudFormation::Authentication:
        S3Auth:
          type: "s3"
          buckets: ["elasticbeanstalk-us-west-2-123456789012"]
          roleName:
            "Fn::GetOptionSetting":
              Namespace: "aws:autoscaling:launchconfiguration"
              OptionName: "IamInstanceProfile"
              DefaultValue: "aws-elasticbeanstalk-ec2-role"
```

Custom resource examples

The following is a list of example configuration files that you can use to customize your Elastic Beanstalk environments:

- DynamoDB, CloudWatch, and SNS
- Elastic Load Balancing and CloudWatch
- ElastiCache
- RDS and CloudWatch
- SQS, SNS, and CloudWatch

Subtopics of this page provide some extended examples for adding and configuring custom resources in an Elastic Beanstalk environment.

Examples

- Example: ElastiCache
- Example: SQS, CloudWatch, and SNS
- Example: DynamoDB, CloudWatch, and SNS

Example: ElastiCache

The following samples add an Amazon ElastiCache cluster to EC2-Classic and EC2-VPC (both default and custom Amazon Virtual Private Cloud (Amazon VPC)) platforms. For more information about these platforms and how you can determine which ones EC2 supports for your region and your AWS account, see https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-supported-platforms.html. Then refer to the section in this topic that applies to your platform.

- EC2-classic platforms
- EC2-VPC (default)
- EC2-VPC (custom)

EC2-classic platforms

This sample adds an Amazon ElastiCache cluster to an environment with instances launched into the EC2-Classic platform. All of the properties that are listed in this example are the minimum required properties that must be set for each resource type. You can download the example at ElastiCache example.

Note

This example creates AWS resources, which you might be charged for. For more information about AWS pricing, see https://aws.amazon.com/pricing/. Some services are part of the AWS Free Usage Tier. If you are a new customer, you can test drive these services for free. See https://aws.amazon.com/free/ for more information.

To use this example, do the following:

1. Create an `.ebextensions` directory in the top-level directory of your source bundle.
2. Create two configuration files with the `.config` extension and place them in your `.ebextensions` directory. One configuration file defines the resources, and the other configuration file defines the options.
3. Deploy your application to Elastic Beanstalk.

   YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

Create a configuration file (e.g., `elasticache.config`) that defines the resources. In this example, we create the ElastiCache cluster by specifying the name of the ElastiCache cluster resource (`MyElastiCache`), declaring its type, and then configuring the properties for the cluster. The example references the name of the ElastiCache security group resource that gets created and defined in this configuration file. Next, we create an ElastiCache security group. We define the name for this resource, declare its type, and add a description for the security group. Finally, we set the ingress rules for the ElastiCache security group to allow access only from instances inside the ElastiCache security group (`MyCacheSecurityGroup`) and the Elastic Beanstalk security group (`AWSEBSecurityGroup`). The parameter name, `AWSEBSecurityGroup`, is a fixed resource name provided by Elastic Beanstalk. You must add `AWSEBSecurityGroup` to your ElastiCache security group ingress rules in order for your Elastic Beanstalk application to connect to the instances in your ElastiCache cluster.

```
#This sample requires you to create a separate configuration file that defines the
 custom option settings for CacheCluster properties.

Resources:
  MyElastiCache:
    Type: AWS::ElastiCache::CacheCluster
    Properties:
      CacheNodeType:
         Fn::GetOptionSetting:
             OptionName : CacheNodeType
             DefaultValue: cache.m1.small
      NumCacheNodes:
           Fn::GetOptionSetting:
             OptionName : NumCacheNodes
             DefaultValue: 1
      Engine:
           Fn::GetOptionSetting:
             OptionName : Engine
             DefaultValue: memcached
      CacheSecurityGroupNames:
        - Ref: MyCacheSecurityGroup
  MyCacheSecurityGroup:
    Type: AWS::ElastiCache::SecurityGroup
    Properties:
      Description: "Lock cache down to webserver access only"
  MyCacheSecurityGroupIngress:
    Type: AWS::ElastiCache::SecurityGroupIngress
    Properties:
      CacheSecurityGroupName:
        Ref: MyCacheSecurityGroup
      EC2SecurityGroupName:
        Ref: AWSEBSecurityGroup
```

For more information about the resources used in this example configuration file, see the following references:

- AWS::ElastiCache::CacheCluster
- AWS::ElastiCache::SecurityGroup
- AWS::ElastiCache:SecurityGroupIngress

Create a separate configuration file called `options.config` and define the custom option settings.

```
option_settings:
  "aws:elasticbeanstalk:customoption":
     CacheNodeType : cache.m1.small
     NumCacheNodes : 1
     Engine : memcached
```

These lines tell Elastic Beanstalk to get the values for the **CacheNodeType, NumCacheNodes, and Engine** properties from the **CacheNodeType, NumCacheNodes, and Engine** values in a config file (options.config in our example) that contains an option_settings section with an **aws:elasticbeanstalk:customoption** section that contains a name-value pair that contains the actual value to use. In the example above, this means cache.m1.small, 1, and memcached would be used for the values. For more information about `Fn::GetOptionSetting`, see Functions.

EC2-VPC (default)

This sample adds an Amazon ElastiCache cluster to an environment with instances launched into the EC2-VPC platform. Specifically, the information in this section applies to a scenario where EC2 launches instances into the default VPC. All of the properties in this example are the minimum required properties that must be set for each resource type. For more information about default VPCs, see Your Default VPC and Subnets.

Note

This example creates AWS resources, which you might be charged for. For more information about AWS pricing, see https://aws.amazon.com/pricing/. Some services are part of the AWS Free Usage Tier. If you are a new customer, you can test drive these services for free. See https://aws.amazon.com/free/ for more information.

To use this example, do the following:

1. Create an `.ebextensions` directory in the top-level directory of your source bundle.
2. Create two configuration files with the `.config` extension and place them in your `.ebextensions` directory. One configuration file defines the resources, and the other configuration file defines the options.
3. Deploy your application to Elastic Beanstalk.

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

Now name the resources configuration file `elasticache.config`. To create the ElastiCache cluster, this example specifies the name of the ElastiCache cluster resource (`MyElastiCache`), declares its type, and then configures the properties for the cluster. The example references the ID of the security group resource that we create and define in this configuration file.

Next, we create an EC2 security group. We define the name for this resource, declare its type, add a description, and set the ingress rules for the security group to allow access only from instances inside the Elastic Beanstalk security group (`AWSEBSecurityGroup`). (The parameter name, `AWSEBSecurityGroup`, is a fixed resource name provided by Elastic Beanstalk. You must add `AWSEBSecurityGroup` to your ElastiCache security group ingress rules in order for your Elastic Beanstalk application to connect to the instances in your ElastiCache cluster.)

The ingress rules for the EC2 security group also define the IP protocol and port numbers on which the cache nodes can accept connections. For Redis, the default port number is `6379`.

```
#This sample requires you to create a separate configuration file that defines the
 custom option settings for CacheCluster properties.

Resources:
  MyCacheSecurityGroup:
    Type: "AWS::EC2::SecurityGroup"
    Properties:
      GroupDescription: "Lock cache down to webserver access only"
      SecurityGroupIngress :
        - IpProtocol : "tcp"
          FromPort :
            Fn::GetOptionSetting:
              OptionName : "CachePort"
              DefaultValue: "6379"
          ToPort :
            Fn::GetOptionSetting:
              OptionName : "CachePort"
              DefaultValue: "6379"
          SourceSecurityGroupName:
            Ref: "AWSEBSecurityGroup"
  MyElastiCache:
    Type: "AWS::ElastiCache::CacheCluster"
    Properties:
      CacheNodeType:
        Fn::GetOptionSetting:
          OptionName : "CacheNodeType"
          DefaultValue : "cache.t2.micro"
      NumCacheNodes:
        Fn::GetOptionSetting:
          OptionName : "NumCacheNodes"
          DefaultValue : "1"
      Engine:
        Fn::GetOptionSetting:
          OptionName : "Engine"
          DefaultValue : "redis"
      VpcSecurityGroupIds:
        -
          Fn::GetAtt:
            - MyCacheSecurityGroup
            - GroupId

Outputs:
  ElastiCache:
    Description : "ID of ElastiCache Cache Cluster with Redis Engine"
    Value :
      Ref : "MyElastiCache"
```

For more information about the resources used in this example configuration file, see the following references:

- AWS::ElastiCache::CacheCluster
- AWS::EC2::SecurityGroup

Next, name the options configuration file `options.config` and define the custom option settings.

```
option_settings:
  "aws:elasticbeanstalk:customoption":
    CacheNodeType : cache.t2.micro
    NumCacheNodes : 1
    Engine : redis
    CachePort : 6379
```

These lines tell Elastic Beanstalk to get the values for the `CacheNodeType`, `NumCacheNodes`, `Engine`, and `CachePort` properties from the `CacheNodeType`, `NumCacheNodes`, `Engine`, and `CachePort` values in a config file (`options.config` in our example). That file includes an `aws:elasticbeanstalk:customoption` section (under `option_settings`) that contains name-value pairs with the actual values to use. In the preceding example, `cache.t2.micro`, 1, `redis`, and `6379` would be used for the values. For more information about `Fn::GetOptionSetting`, see Functions.

EC2-VPC (custom)

If you create a custom VPC on the EC2-VPC platform and specify it as the VPC into which EC2 launches instances, the process of adding an Amazon ElastiCache cluster to your environment differs from that of a default VPC. The main difference is that you must create a subnet group for the ElastiCache cluster. All of the properties in this example are the minimum required properties that must be set for each resource type.

Note

This example creates AWS resources, which you might be charged for. For more information about AWS pricing, see https://aws.amazon.com/pricing/. Some services are part of the AWS Free Usage Tier. If you are a new customer, you can test drive these services for free. See https://aws.amazon.com/free/ for more information.

To use this example, do the following:

1. Create an `.ebextensions` directory in the top-level directory of your source bundle.
2. Create two configuration files with the `.config` extension and place them in your `.ebextensions` directory. One configuration file defines the resources, and the other configuration file defines the options.
3. Deploy your application to Elastic Beanstalk.

   YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

Now name the resources configuration file `elasticache.config`. To create the ElastiCache cluster, this example specifies the name of the ElastiCache cluster resource (`MyElastiCache`), declares its type, and then configures the properties for the cluster. The properties in the example reference the name of the subnet group for the ElastiCache cluster as well as the ID of security group resource that we create and define in this configuration file.

Next, we create an EC2 security group. We define the name for this resource, declare its type, add a description, the VPC ID, and set the ingress rules for the security group to allow access only from instances inside the Elastic Beanstalk security group (`AWSEBSecurityGroup`). (The parameter name, `AWSEBSecurityGroup`, is a fixed resource name provided by Elastic Beanstalk. You must add `AWSEBSecurityGroup` to your ElastiCache security group ingress rules in order for your Elastic Beanstalk application to connect to the instances in your ElastiCache cluster.)

The ingress rules for the EC2 security group also define the IP protocol and port numbers on which the cache nodes can accept connections. For Redis, the default port number is `6379`. Finally, this example creates a subnet group for the ElastiCache cluster. We define the name for this resource, declare its type, and add a description and ID of the subnet in the subnet group.

Note

We recommend that you use private subnets for the ElastiCache cluster. For more information about a VPC with a private subnet, see https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Scenario2.html.

```
#This sample requires you to create a separate configuration file that defines the
 custom option settings for CacheCluster properties.

Resources:
  MyElastiCache:
    Type: "AWS::ElastiCache::CacheCluster"
    Properties:
      CacheNodeType:
        Fn::GetOptionSetting:
          OptionName : "CacheNodeType"
          DefaultValue : "cache.t2.micro"
      NumCacheNodes:
        Fn::GetOptionSetting:
          OptionName : "NumCacheNodes"
          DefaultValue : "1"
      Engine:
        Fn::GetOptionSetting:
          OptionName : "Engine"
          DefaultValue : "redis"
      CacheSubnetGroupName:
        Ref: "MyCacheSubnets"
      VpcSecurityGroupIds:
        - Ref: "MyCacheSecurityGroup"
  MyCacheSecurityGroup:
    Type: "AWS::EC2::SecurityGroup"
    Properties:
      GroupDescription: "Lock cache down to webserver access only"
      VpcId:
        Fn::GetOptionSetting:
          OptionName : "VpcId"
      SecurityGroupIngress :
        - IpProtocol : "tcp"
          FromPort :
            Fn::GetOptionSetting:
              OptionName : "CachePort"
              DefaultValue: "6379"
          ToPort :
            Fn::GetOptionSetting:
              OptionName : "CachePort"
              DefaultValue: "6379"
          SourceSecurityGroupId:
            Ref: "AWSEBSecurityGroup"
  MyCacheSubnets:
    Type: "AWS::ElastiCache::SubnetGroup"
    Properties:
      Description: "Subnets for ElastiCache"
      SubnetIds:
        Fn::GetOptionSetting:
          OptionName : "CacheSubnets"
Outputs:
  ElastiCache:
    Description : "ID of ElastiCache Cache Cluster with Redis Engine"
    Value :
      Ref : "MyElastiCache"
```

For more information about the resources used in this example configuration file, see the following references:

- AWS::ElastiCache::CacheCluster
- AWS::EC2::SecurityGroup

- AWS::ElastiCache::SubnetGroup

Next, name the options configuration file `options.config` and define the custom option settings.

Note

In the following example, replace the example `CacheSubnets` and `VpcId` values with your own subnets and VPC.

```
option_settings:
  "aws:elasticbeanstalk:customoption":
    CacheNodeType : cache.t2.micro
    NumCacheNodes : 1
    Engine : redis
    CachePort : 6379
    CacheSubnets:
      - subnet-1a1a1a1a
      - subnet-2b2b2b2b
      - subnet-3c3c3c3c
    VpcId: vpc-4d4d4d4d
```

These lines tell Elastic Beanstalk to get the values for the `CacheNodeType`, `NumCacheNodes`, `Engine`, `CachePort`, `CacheSubnets`, and `VpcId` properties from the `CacheNodeType`, `NumCacheNodes`, `Engine`, `CachePort`, `CacheSubnets`, and `VpcId` values in a config file (`options.config` in our example). That file includes an `aws:elasticbeanstalk:customoption` section (under `option_settings`) that contains name-value pairs with sample values. In the example above, `cache.t2.micro`, 1, `redis`, 6379, `subnet-1a1a1a1a`, `subnet-2b2b2b2b`, `subnet-3c3c3c3c`, and `vpc-4d4d4d4d` would be used for the values. For more information about `Fn::GetOptionSetting`, see Functions.

Example: SQS, CloudWatch, and SNS

This example adds an Amazon SQS queue and an alarm on queue depth to the environment. The properties that you see in this example are the minimum required properties that you must set for each of these resources. You can download the example at SQS, SNS, and CloudWatch.

Note

This example creates AWS resources, which you might be charged for. For more information about AWS pricing, see https://aws.amazon.com/pricing/. Some services are part of the AWS Free Usage Tier. If you are a new customer, you can test drive these services for free. See https://aws.amazon.com/free/ for more information.

To use this example, do the following:

1. Create an `.ebextensions` directory in the top-level directory of your source bundle.
2. Create two configuration files with the `.config` extension and place them in your `.ebextensions` directory. One configuration file defines the resources, and the other configuration file defines the options.
3. Deploy your application to Elastic Beanstalk.

   YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

Create a configuration file (e.g., sqs.config) that defines the resources. In this example, we create an SQS queue and define the `VisbilityTimeout` property in the `MySQSQueue` resource. Next, we create an SNS `Topic` and specify that email gets sent to `someone@example.com` when the alarm is fired. Finally, we create a CloudWatch alarm if the queue grows beyond 10 messages. In the `Dimensions` property, we specify the name of the dimension and the value representing the dimension measurement. We use `Fn::GetAtt` to return the value of `QueueName` from `MySQSQueue`.

```
#This sample requires you to create a separate configuration file to define the custom
 options for the SNS topic and SQS queue.
Resources:
  MySQSQueue:
    Type: AWS::SQS::Queue
    Properties:
      VisibilityTimeout:
        Fn::GetOptionSetting:
          OptionName: VisibilityTimeout
          DefaultValue: 30
  AlarmTopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Endpoint:
            Fn::GetOptionSetting:
              OptionName: AlarmEmail
              DefaultValue: "nobody@amazon.com"
          Protocol: email
  QueueDepthAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmDescription: "Alarm if queue depth grows beyond 10 messages"
      Namespace: "AWS/SQS"
      MetricName: ApproximateNumberOfMessagesVisible
      Dimensions:
        - Name: QueueName
          Value : { "Fn::GetAtt" : [ "MySQSQueue", "QueueName"] }
      Statistic: Sum
      Period: 300
      EvaluationPeriods: 1
      Threshold: 10
      ComparisonOperator: GreaterThanThreshold
      AlarmActions:
        - Ref: AlarmTopic
      InsufficientDataActions:
        - Ref: AlarmTopic

Outputs :
  QueueURL:
    Description : "URL of newly created SQS Queue"
    Value : { Ref : "MySQSQueue" }
  QueueARN :
    Description : "ARN of newly created SQS Queue"
    Value : { "Fn::GetAtt" : [ "MySQSQueue", "Arn"]}
  QueueName :
    Description : "Name newly created SQS Queue"
    Value : { "Fn::GetAtt" : [ "MySQSQueue", "QueueName"]}
```

For more information about the resources used in this example configuration file, see the following references:

- AWS::SQS::Queue

- AWS::SNS::Topic
- AWS::CloudWatch::Alarm

Create a separate configuration file called `options.config` and define the custom option settings.

```
option_settings:
  "aws:elasticbeanstalk:customoption":
     VisibilityTimeout : 30
     AlarmEmail : "nobody@example.com"
```

These lines tell Elastic Beanstalk to get the values for the **VisibilityTimeout and Subscription Endpoint** properties from the **VisibilityTimeout and Subscription Endpoint** values in a config file (options.config in our example) that contains an option_settings section with an **aws:elasticbeanstalk:customoption** section that contains a name-value pair that contains the actual value to use. In the example above, this means 30 and "nobody@amazon.com" would be used for the values. For more information about `Fn::GetOptionSetting`, see the section called "Functions".

Example: DynamoDB, CloudWatch, and SNS

This configuration file sets up the DynamoDB table as a session handler for a PHP-based application using the AWS SDK for PHP 2. To use this example, you must have an IAM instance profile, which is added to the instances in your environment and used to access the DynamoDB table.

You can download the sample that we'll use in this step at DynamoDB session Support example. The sample contains the following files:

- The sample application, `index.php`
- A configuration file, `dynamodb.config`, to create and configure a DynamoDB table and other AWS resources and install software on the EC2 instances that host the application in an Elastic Beanstalk environment
- A configuration file, `options.config`, that overrides the defaults in `dynamodb.config` with specific settings for this particular installation

`index.php`

```
<?php

// Include the SDK using the Composer autoloader
require '../vendor/autoload.php';

use Aws\DynamoDb\DynamoDbClient;

// Grab the session table name and region from the configuration file
list($tableName, $region) = file(__DIR__ . '/../sessiontable');
$tableName = rtrim($tableName);
$region = rtrim($region);

// Create a DynamoDB client and register the table as the session handler
$dynamodb = DynamoDbClient::factory(array('region' => $region));
$handler = $dynamodb->registerSessionHandler(array('table_name' => $tableName,
 'hash_key' => 'username'));

// Grab the instance ID so we can display the EC2 instance that services the request
$instanceId = file_get_contents("http://169.254.169.254/latest/meta-data/instance-id");
?>
<h1>Elastic Beanstalk PHP Sessions Sample</h1>
<p>This sample application shows the integration of the Elastic Beanstalk PHP
container and the session support for DynamoDB from the AWS SDK for PHP 2.
Using DynamoDB session support, the application can be scaled out across
multiple web servers. For more details, see the
<a href="https://aws.amazon.com/php/">PHP Developer Center</a>.</p>

<form id="SimpleForm" name="SimpleForm" method="post" action="index.php">
<?php
echo 'Request serviced from instance ' . $instanceId . '<br/>';
echo '<br/>';

if (isset($_POST['continue'])) {
  session_start();
  $_SESSION['visits'] = $_SESSION['visits'] + 1;
  echo 'Welcome back ' . $_SESSION['username'] . '<br/>';
  echo 'This is visit number ' . $_SESSION['visits'] . '<br/>';
  session_write_close();
  echo '<br/>';
  echo '<input type="Submit" value="Refresh" name="continue" id="continue"/>';
  echo '<input type="Submit" value="Delete Session" name="killsession"
 id="killsession"/>';
} elseif (isset($_POST['killsession'])) {
  session_start();
  echo 'Goodbye ' . $_SESSION['username'] . '<br/>';
  session_destroy();
  echo 'Username: <input type="text" name="username" id="username" size="30"/><br/>';
  echo '<br/>';
  echo '<input type="Submit" value="New Session" name="newsession" id="newsession"/>';
} elseif (isset($_POST['newsession'])) {
  session_start();
  $_SESSION['username'] = $_POST['username'];
  $_SESSION['visits'] = 1;
  echo 'Welcome to a new session ' . $_SESSION['username'] . '<br/>';
  session_write_close();
  echo '<br/>';
  echo '<input type="Submit" value="Refresh" name="continue" id="continue"/>';
  echo '<input type="Submit" value="Delete Session" name="killsession"
 id="killsession"/>';
} else {
  echo 'To get started, enter a username.<br/>';
  echo '<br/>';
  echo 'Username: <input type="text" name="username" id="username" size="30"/><br/>';
  echo '<input type="Submit" value="New Session" name="newsession" id="newsession"/>';
}
?>
</form>
```

.ebextensions/dynamodb.config

```
Resources:
  SessionTable:
    Type: AWS::DynamoDB::Table
    Properties:
      KeySchema:
        HashKeyElement:
          AttributeName:
            Fn::GetOptionSetting:
              OptionName : SessionHashKeyName
              DefaultValue: "username"
          AttributeType:
            Fn::GetOptionSetting:
              OptionName : SessionHashKeyType
              DefaultValue: "S"
      ProvisionedThroughput:
        ReadCapacityUnits:
          Fn::GetOptionSetting:
            OptionName : SessionReadCapacityUnits
            DefaultValue: 1
        WriteCapacityUnits:
          Fn::GetOptionSetting:
            OptionName : SessionWriteCapacityUnits
            DefaultValue: 1

  SessionWriteCapacityUnitsLimit:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmDescription: { "Fn::Join" : ["", [{ "Ref" : "AWSEBEnvironmentName" }, "
 write capacity limit on the session table." ]]}
      Namespace: "AWS/DynamoDB"
      MetricName: ConsumedWriteCapacityUnits
      Dimensions:
        - Name: TableName
          Value: { "Ref" : "SessionTable" }
      Statistic: Sum
      Period: 300
      EvaluationPeriods: 12
      Threshold:
          Fn::GetOptionSetting:
            OptionName : SessionWriteCapacityUnitsAlarmThreshold
            DefaultValue: 240
      ComparisonOperator: GreaterThanThreshold
      AlarmActions:
        - Ref: SessionAlarmTopic
      InsufficientDataActions:
        - Ref: SessionAlarmTopic

  SessionReadCapacityUnitsLimit:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmDescription: { "Fn::Join" : ["", [{ "Ref" : "AWSEBEnvironmentName" }, " read
 capacity limit on the session table." ]]}
      Namespace: "AWS/DynamoDB"
      MetricName: ConsumedReadCapacityUnits
      Dimensions:
        - Name: TableName
          Value: { "Ref" : "SessionTable" }
      Statistic: Sum
      Period: 300
      EvaluationPeriods: 12
      Threshold:
          Fn::GetOptionSetting:
            OptionName : SessionReadCapacityUnitsAlarmThreshold
            DefaultValue: 240
      ComparisonOperator: GreaterThanThreshold
      AlarmActions:
        - Ref: SessionAlarmTopic
      InsufficientDataActions:
        - Ref: SessionAlarmTopic

  SessionThrottledRequestsAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmDescription: { "Fn::Join" : ["", [{ "Ref" : "AWSEBEnvironmentName" }, ":
 requests are being throttled." ]]}
      Namespace: AWS/DynamoDB
      MetricName: ThrottledRequests
      Dimensions:
        - Name: TableName
          Value: { "Ref" : "SessionTable" }
      Statistic: Sum
      Period: 300
      EvaluationPeriods: 1
      Threshold:
        Fn::GetOptionSetting:
          OptionName: SessionThrottledRequestsThreshold
          DefaultValue: 1
      ComparisonOperator: GreaterThanThreshold
      AlarmActions:
        - Ref: SessionAlarmTopic
      InsufficientDataActions:
        - Ref: SessionAlarmTopic

  SessionAlarmTopic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Endpoint:
            Fn::GetOptionSetting:
              OptionName: SessionAlarmEmail
              DefaultValue: "nobody@amazon.com"
          Protocol: email

files:
  "/var/app/sessiontable":
    mode: "000444"
    content: |
      `{"Ref" : "SessionTable"}`
      `{"Ref" : "AWS::Region"}`

  "/var/app/composer.json":
    mode: "000744"
    content:
      {
        "require": {
           "aws/aws-sdk-php": "*"
        }
      }

container_commands:
 "1-install-composer":
   command: "cd /var/app; curl -s http://getcomposer.org/installer | php"
 "2-install-dependencies":
   command: "cd /var/app; php composer.phar install"
 "3-cleanup-composer":
   command: "rm -Rf /var/app/composer.*"
```

In the sample configuration file, we first create the DynamoDB table and configure the primary key structure for the table and the capacity units to allocate sufficient resources to provide the requested throughput. Next, we create CloudWatch alarms for `WriteCapacity` and `ReadCapacity`. We create an SNS topic that sends email to "nobody@amazon.com" if the alarm thresholds are breached.

After we create and configure our AWS resources for our environment, we need to customize the EC2 instances. We use the `files` key to pass the details of the DynamoDB table to the EC2 instances in our environment as well as add a "require" in the `composer.json` file for the AWS SDK for PHP 2. Finally, we run container commands to install composer, the required dependencies, and then remove the installer.

.ebextensions/options.config

```
option_settings:
  "aws:elasticbeanstalk:customoption":
     SessionHashKeyName                      : username
     SessionHashKeyType                      : S
     SessionReadCapacityUnits                : 1
```

```
      SessionReadCapacityUnitsAlarmThreshold  : 240
      SessionWriteCapacityUnits               : 1
      SessionWriteCapacityUnitsAlarmThreshold : 240
      SessionThrottledRequestsThreshold       : 1
      SessionAlarmEmail                       : me@example.com
```

Replace the SessionAlarmEmail value with the email where you want alarm notifications sent. The `options.config` file contains the values used for some of the variables defined in `dynamodb.config`. For example, `dynamodb.config` contains the following lines:

```
Subscription:
  - Endpoint:
      Fn::GetOptionSetting:
        OptionName: SessionAlarmEmail
        DefaultValue: "nobody@amazon.com"
```

These lines that tell Elastic Beanstalk to get the value for the **Endpoint** property from the **SessionAlarmEmail** value in a config file (`options.config` in our sample application) that contains an option_settings section with an **aws:elasticbeanstalk:customoption** section that contains a name-value pair that contains the actual value to use. In the example above, this means **SessionAlarmEmail** would be assigned the value `nobody@amazon.com`.

For more information about the CloudFormation resources used in this example, see the following references:

- AWS::DynamoDB::Table
- AWS::CloudWatch::Alarm
- AWS::SNS::Topic

Using Elastic Beanstalk saved configurations

You can save your environment's configuration as an object in Amazon Simple Storage Service (Amazon S3) that can be applied to other environments during environment creation, or applied to a running environment. *Saved configurations* are YAML formatted templates that define an environment's platform version, tier, configuration option settings, and tags.

You can apply tags to a saved configuration when you create it, and edit tags of existing saved configurations. The tags applied to a saved configuration aren't related to the tags specified in a saved configuration using the `Tags:` key. The latter are applied to an environment when you apply the saved configuration to the environment. For details, see Tagging saved configurations.

Note

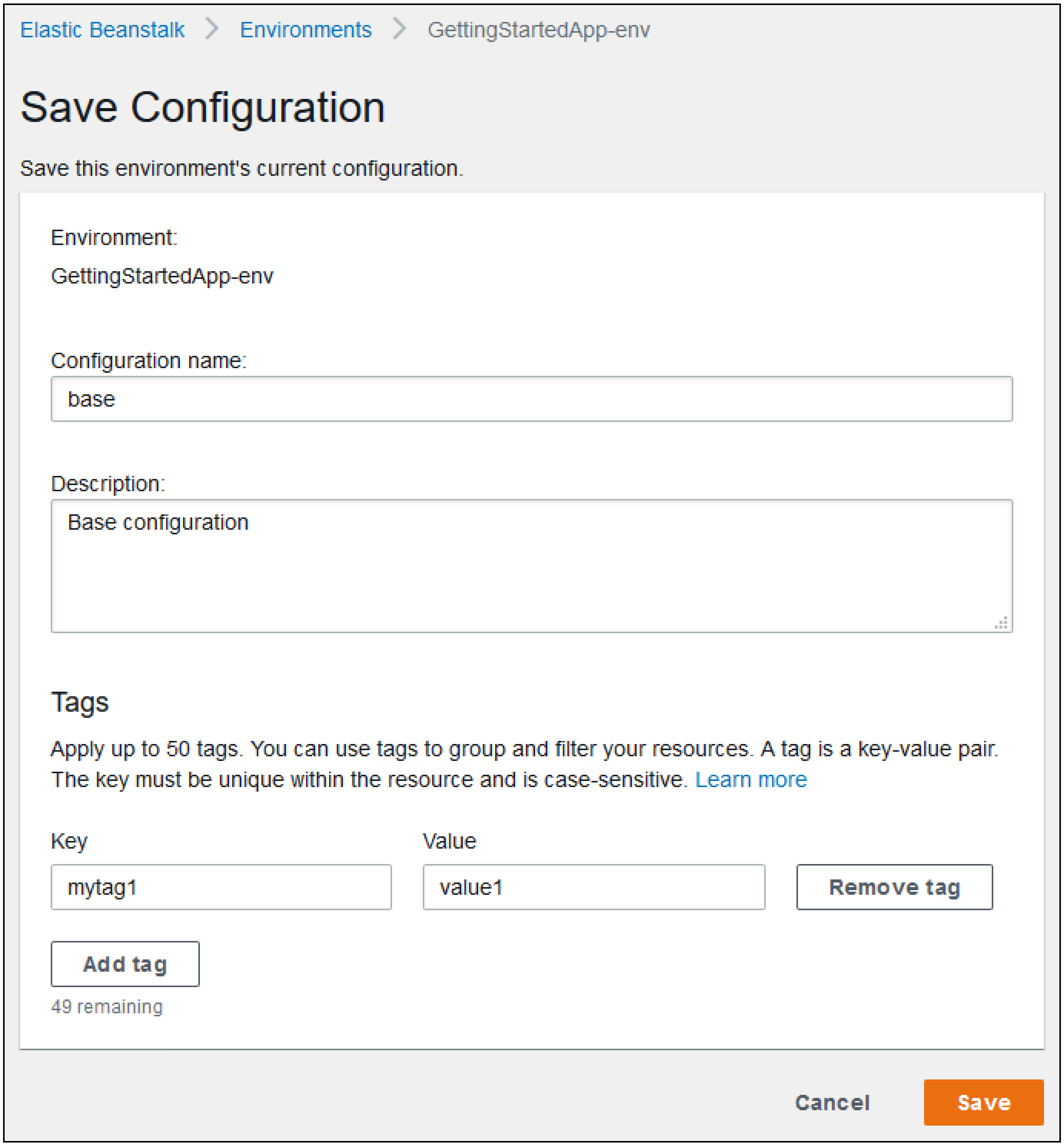
You can create and apply saved configurations to your Elastic Beanstalk environments using several methods. These include the Elastic Beanstalk console, the EB CLI, and the AWS CLI. See the following topics for examples of alternate methods for creating and applying saved configurations:

- Setting configuration options before environment creation
- Setting configuration options during environment creation
- Setting configuration options after environment creation

Create a saved configuration from the current state of your environment in the Elastic Beanstalk management console.

To save an environment's configuration

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Save configuration**.
4. Use the on-screen form to name the saved configuration. Optionally, provide a brief description, and add tag keys and values.
5. Choose **Save**.



Save Configuration

Save this environment's current configuration.

Environment:
GettingStartedApp-env

Configuration name: base

Description: Base configuration

Tags

Apply up to 50 tags. You can use tags to group and filter your resources. A tag is a key-value pair. The key must be unique within the resource and is case-sensitive. Learn more

Key	Value
mytag1	value1

The saved configuration includes any settings that you have applied to the environment with the console or any other client that uses the Elastic Beanstalk API. You can then apply the saved configuration to your environment at a later date to restore it to its previous state, or apply it to a new environment during environment creation.

You can download a configuration using the EB CLI the section called “**eb config**” command, as shown in the following example. *NAME* is the name of your saved configuration.

```
eb config get NAME
```

To apply a saved configuration during environment creation (Elastic Beanstalk console)

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.
3. In the navigation pane, find your application's name and choose **Saved configurations**.
4. Select the saved configuration you want to apply, and then choose **Launch environment**.
5. Proceed through the wizard to create your environment.

Saved configurations don't include settings applied with configuration files in your application's source code. If the same setting is applied in both a configuration file and saved configuration, the setting in the saved configuration takes precedence. Likewise, options specified in the Elastic Beanstalk console override options in saved configurations. For more information, see Precedence.

Saved configurations are stored in the Elastic Beanstalk S3 bucket in a folder named after your application. For example, configurations for an application named `my-app` in the us-west-2 region for account number 123456789012 can be found at `s3://elasticbeanstalk-us-west-2-123456789012/resources/templates/my-app/`.

View the contents of a saved configuration by opening it in a text editor. The following example configuration shows the configuration of a web server environment launched with the Elastic Beanstalk management console.

```
EnvironmentConfigurationMetadata:
  Description: Saved configuration from a multicontainer Docker environment created
 with the Elastic Beanstalk Management Console
  DateCreated: '1520633151000'
  DateModified: '1520633151000'
Platform:
  PlatformArn: arn:aws:elasticbeanstalk:us-east-2::platform/Java 8 running on 64bit
 Amazon Linux/2.5.0
OptionSettings:
  aws:elasticbeanstalk:command:
    BatchSize: '30'
    BatchSizeType: Percentage
  aws:elasticbeanstalk:sns:topics:
    Notification Endpoint: me@example.com
  aws:elb:policies:
    ConnectionDrainingEnabled: true
    ConnectionDrainingTimeout: '20'
  aws:elb:loadbalancer:
    CrossZone: true
  aws:elasticbeanstalk:environment:
    ServiceRole: aws-elasticbeanstalk-service-role
  aws:elasticbeanstalk:application:
    Application Healthcheck URL: /
  aws:elasticbeanstalk:healthreporting:system:
    SystemType: enhanced
  aws:autoscaling:launchconfiguration:
    IamInstanceProfile: aws-elasticbeanstalk-ec2-role
    InstanceType: t2.micro
    EC2KeyName: workstation-uswest2
  aws:autoscaling:updatepolicy:rollingupdate:
    RollingUpdateType: Health
    RollingUpdateEnabled: true
EnvironmentTier:
  Type: Standard
  Name: WebServer
AWSConfigurationTemplateVersion: 1.1.0.0
Tags:
  Cost Center: WebApp Dev
```

You can modify the contents of a saved configuration and save it in the same location in Amazon S3. Any properly formatted saved configuration stored in the right location can be applied to an environment by using the Elastic Beanstalk management console.

The following keys are supported.

- **AWSConfigurationTemplateVersion** (required) – The configuration template version (1.1.0.0).

  ```
  AWSConfigurationTemplateVersion: 1.1.0.0
  ```

- **Platform** – The Amazon Resource Name (ARN) of the environment's platform version. You can specify the platform by ARN or solution stack name.

  ```
  Platform:
  ```

```
  PlatformArn: arn:aws:elasticbeanstalk:us-east-2::platform/Java 8 running on 64bit
 Amazon Linux/2.5.0
```

- **SolutionStack** – The full name of the solution stack used to create the environment.

```
SolutionStack: 64bit Amazon Linux 2017.03 v2.5.0 running Java 8
```

- **OptionSettings** – Configuration option settings to apply to the environment. For example, the following entry sets the instance type to t2.micro.

```
OptionSettings:
  aws:autoscaling:launchconfiguration:
    InstanceType: t2.micro
```

- **Tags** – Up to 47 tags to apply to resources created within the environment.

```
Tags:
  Cost Center: WebApp Dev
```

- **EnvironmentTier** – The type of environment to create. For a web server environment, you can exclude this section (web server is the default). For a worker environment, use the following.

```
EnvironmentTier:
  Name: Worker
  Type: SQS/HTTP
```

Note

You can create and apply saved configurations to your Elastic Beanstalk environments using several methods. These include the Elastic Beanstalk console, the EB CLI, and the AWS CLI. See the following topics for examples of alternate methods for creating and applying saved configurations:

- Setting configuration options before environment creation
- Setting configuration options during environment creation
- Setting configuration options after environment creation

Tagging saved configurations

You can apply tags to your AWS Elastic Beanstalk saved configurations. Tags are key-value pairs associated with AWS resources. For information about Elastic Beanstalk resource tagging, use cases, tag key and value constraints, and supported resource types, see Tagging Elastic Beanstalk application resources.

You can specify tags when you create a saved configuration. In an existing saved configuration, you can add or remove tags, and update the values of existing tags. You can add up to 50 tags to each saved configuration.

Adding tags during saved configuration creation

When you use the Elastic Beanstalk console to save a configuration, you can specify tag keys and values on the **Save Configuration** page.

If you use the EB CLI to save a configuration, use the `--tags` option with **eb config** to add tags.

```
~/workspace/my-app$ eb config --tags mytag1=value1,mytag2=value2
```

With the AWS CLI or other API-based clients, add tags by using the `--tags` parameter on the **create-configuration-template** command.

```
$ aws elasticbeanstalk create-configuration-template \
      --tags Key=mytag1,Value=value1 Key=mytag2,Value=value2 \
      --application-name my-app --template-name my-template --solution-stack-
name solution-stack
```

Managing tags of an existing saved configuration

You can add, update, and delete tags in an existing Elastic Beanstalk saved configuration.

To manage a saved configuration's tags using the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.
3. In the navigation pane, find your application's name and choose **Saved configurations**.

4. Select the saved configuration you want to manage.
5. Choose **Actions**, and then choose **Manage tags**.
6. Use the on-screen form to add, update, or delete tags.
7. To save the changes choose **Apply** at the bottom of the page.

If you use the EB CLI to update your saved configuration, use **eb tags** to add, update, delete, or list tags.

For example, the following command lists the tags in a saved configuration.

```
~/workspace/my-app$ eb tags --list --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:configurationtemplate/my-app/my-template"
```

The following command updates the tag `mytag1` and deletes the tag `mytag2`.

```
~/workspace/my-app$ eb tags --update mytag1=newvalue --delete mytag2 \
      --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:configurationtemplate/my-app/my-template"
```

For a complete list of options and more examples, see `eb tags`.

With the AWS CLI or other API-based clients, use the **list-tags-for-resource** command to list the tags of a saved configuration.

```
$ aws elasticbeanstalk list-tags-for-resource --resource-arn
 "arn:aws:elasticbeanstalk:us-east-2:my-account-id:configurationtemplate/my-app/my-template"
```

Use the **update-tags-for-resource** command to add, update, or delete tags in a saved configuration.

```
$ aws elasticbeanstalk update-tags-for-resource \
      --tags-to-add Key=mytag1,Value=newvalue --tags-to-remove mytag2 \
      --resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-id:configurationtemplate/my-app/my-template"
```

Specify both tags to add and tags to update in the `--tags-to-add` parameter of **update-tags-for-resource**. A nonexisting tag is added, and an existing tag's value is updated.

Note

To use some of the EB CLI and AWS CLI commands with an Elastic Beanstalk saved configuration, you need the saved configuration's ARN. To construct the ARN, first use the following command to retrieve the saved configuration's name.

```
$ aws elasticbeanstalk describe-applications --application-names my-app
```

Look for the `ConfigurationTemplates` key in the command's output. This element shows the saved configuration's name. Use this name where *`my-template`* is specified in the commands mentioned on this page.

Environment manifest (env.yaml)

You can include a YAML formatted environment manifest in the root of your application source bundle to configure the environment name, solution stack and environment links to use when creating your environment.

This file format includes support for environment groups. To use groups, specify the environment name in the manifest with a + symbol at the end. When you create or update the environment, specify the group name with `--group-name` (AWS CLI) or `--env-group-suffix` (EB CLI). For more information on groups, see Creating and updating groups of Elastic Beanstalk environments.

The following example manifest defines a web server environment with a link to a worker environment component that it is dependent upon. The manifest uses groups to allow creating multiple environments with the same source bundle:

~/myapp/frontend/env.yaml

```
AWSConfigurationTemplateVersion: 1.1.0.0
SolutionStack: 64bit Amazon Linux 2015.09 v2.0.6 running Multi-container Docker 1.7.1
 (Generic)
OptionSettings:
  aws:elasticbeanstalk:command:
    BatchSize: '30'
    BatchSizeType: Percentage
  aws:elasticbeanstalk:sns:topics:
    Notification Endpoint: me@example.com
  aws:elb:policies:
    ConnectionDrainingEnabled: true
    ConnectionDrainingTimeout: '20'
  aws:elb:loadbalancer:
    CrossZone: true
  aws:elasticbeanstalk:environment:
    ServiceRole: aws-elasticbeanstalk-service-role
  aws:elasticbeanstalk:application:
    Application Healthcheck URL: /
  aws:elasticbeanstalk:healthreporting:system:
    SystemType: enhanced
  aws:autoscaling:launchconfiguration:
    IamInstanceProfile: aws-elasticbeanstalk-ec2-role
    InstanceType: t2.micro
    EC2KeyName: workstation-uswest2
  aws:autoscaling:updatepolicy:rollingupdate:
    RollingUpdateType: Health
    RollingUpdateEnabled: true
Tags:
  Cost Center: WebApp Dev
CName: front-A08G28LG+
EnvironmentName: front+
EnvironmentLinks:
  "WORKERQUEUE" : "worker+"
```

The following keys are supported.

- **AWSConfigurationTemplateVersion** (required) – The configuration template version (1.1.0.0).

  ```
  AWSConfigurationTemplateVersion: 1.1.0.0
  ```

- **Platform** – The Amazon Resource Name (ARN) of the environment's platform version. You can specify the platform by ARN or solution stack name.

  ```
  Platform:
    PlatformArn: arn:aws:elasticbeanstalk:us-east-2::platform/Java 8 running on 64bit Amazon Linux/2.5.0
  ```

- **SolutionStack** – The full name of the solution stack used to create the environment.

  ```
  SolutionStack: 64bit Amazon Linux 2017.03 v2.5.0 running Java 8
  ```

- **OptionSettings** – Configuration option settings to apply to the environment. For example, the following entry sets the instance type to t2.micro.

```
OptionSettings:
  aws:autoscaling:launchconfiguration:
    InstanceType: t2.micro
```

- **Tags** – Up to 47 tags to apply to resources created within the environment.

```
Tags:
  Cost Center: WebApp Dev
```

- **EnvironmentTier** – The type of environment to create. For a web server environment, you can exclude this section (web server is the default). For a worker environment, use the following.

```
EnvironmentTier:
  Name: Worker
  Type: SQS/HTTP
```

- **CName** – The CNAME for the environment. Include a + character at the end of the name to enable groups.

```
CName: front-A08G28LG+
```

- **EnvironmentName** – The name of the environment to create. Include a + character at the end of the name to enable groups.

```
EnvironmentName: front+
```

  With groups enabled, you must specify a group name when you create the environments. Elastic Beanstalk appends the group name to the environment name with a hyphen. For example, with the environment name `front+` and the group name `dev`, Elastic Beanstalk will create the environment with the name `front-dev`.

- **EnvironmentLinks** – A map of variable names and environment names of dependencies. The following example makes the `worker+` environment a dependency and tells Elastic Beanstalk to save the link information to a variable named `WORKERQUEUE`.

```
EnvironmentLinks:
  "WORKERQUEUE" : "worker+"
```

The value of the link variable varies depending on the type of the linked environment. For a web server environment, the link is the environment's CNAME. For a worker environment, the link is the name of the environment's Amazon Simple Queue Service (Amazon SQS) queue.

The **CName**, **EnvironmentName** and **EnvironmentLinks** keys can be used to create environment groups and links to other environments. These features are currently supported when using the EB CLI, AWS CLI or an SDK.

Using a custom Amazon machine image (AMI) in your Elastic Beanstalk environment

This section explains when to consider using a custom AMI and provides the procedures to configure and manage the custom AMI in your environment. When you create an AWS Elastic Beanstalk environment, you can specify an Amazon Machine Image (AMI) to use instead of the standard Elastic Beanstalk AMI included in your platform version. A custom AMI can improve provisioning times when instances are launched in your environment if you need to install a lot of software that isn't included in the standard AMIs.

The use of configuration files is effective to customize your environment quickly and consistently. Although applying configurations can start to take a long time during environment creation and updates. If you do a lot of server configuration in configuration files, you can reduce this time by making a custom AMI that already has the software and configuration that you need.

A custom AMI also allows you to make changes to low-level components, such as the Linux kernel, that are difficult to implement or take a long time to apply in configuration files. To create a custom AMI, launch an Elastic Beanstalk platform AMI in Amazon EC2, customize the software and configuration to your needs, and then stop the instance and save an AMI from it.

Creating a custom AMI

You can use EC2 Image Builder to create and manage custom AMIs as an alternative to these procedures. For more information, see the Image Builder User Guide.

To identify the base Elastic Beanstalk AMI

1. In a command window, run a command like the following. For more information, see describe-platform-version in the *AWS CLI Command Reference*.

Specify the AWS Region where you want to use your custom AMI, and replace the platform ARN and version number with the Elastic Beanstalk platform that your application is based on.

Example - Mac OS / Linux OS

```
$ aws elasticbeanstalk describe-platform-version --region us-east-2 \
      --platform-arn "arn:aws:elasticbeanstalk:us-east-2::platform/Node.js 20
 running on 64bit Amazon Linux 2023/6.1.7" \
      --query PlatformDescription.CustomAmiList
[
    {
        "VirtualizationType": "pv",
        "ImageId": ""
    },
    {
        "VirtualizationType": "hvm",
        "ImageId": "ami-020ae06fdda6a0f66"
    }
]
```

Example - Windows OS

```
C:\> aws elasticbeanstalk describe-platform-version --region us-east-2 --platform-
arn"arn:aws:elasticbeanstalk:us-east-2::platform/
IIS 10.0 running on 64bit Windows Server 2022/2.15.3" --query
 PlatformDescription.CustomAmiList
[
    {
        "VirtualizationType": "pv",
        "ImageId": ""
    },
    {
        "VirtualizationType": "hvm",
        "ImageId": "ami-020ae06fdda6a0f66"
    }
]
```

2. Take note of the `ImageId` value that looks like `ami-020ae06fdda6a0f66` in the result.

The value is the stock Elastic Beanstalk AMI for the platform version, EC2 instance architecture, and AWS Region that are relevant for your application. If you need to create AMIs for multiple platforms, architectures or AWS Regions, repeat this process to identify the correct base AMI for each combination.

Note

Don't create an AMI from an instance that has been launched in an Elastic Beanstalk environment. Elastic Beanstalk makes changes to instances during provisioning that can cause issues in the saved AMI. Saving an image from an instance in an Elastic Beanstalk environment will also make the version of your application that was deployed to the instance a fixed part of the image.

For Linux, it is also possible to create a custom AMI from a community AMI that wasn't published by Elastic Beanstalk. You can use the latest Amazon Linux AMI as a starting point. When you launch an environment with a Linux AMI that isn't managed by Elastic Beanstalk, Elastic Beanstalk attempts to install platform software (language, framework, proxy server, etc.) and additional components to support features such as Enhanced Health Reporting.

Note

Custom AMIs based on Windows Server require the stock Elastic Beanstalk AMI returned from `describe-platform-version`, as shown earlier in Step 1.

Although Elastic Beanstalk can use an AMI that isn't managed by Elastic Beanstalk, the increase in provisioning time that results from Elastic Beanstalk installing missing components can reduce or eliminate the benefits of creating a custom AMI in the first place. Other Linux distributions might work with some troubleshooting but are not officially supported. If your application requires a specific Linux distribution, one alternative is to create a Docker image and run it on the Elastic Beanstalk Docker platform or Multicontainer Docker platform.

To create a custom AMI

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose **Launch Instance**.

3. If you identified a base Elastic Beanstalk AMI (using `describe-platform-version`) or an Amazon Linux AMI, enter its AMI ID in the search box. Then press **Enter**.

   You can also search the list for another community AMI that suits your needs.

   > **Note**
   >
   > We recommend that you choose an AMI that uses HVM virtualization. These AMIs show **Virtualization type: hvm** in their description.
   > For more information, see Virtualization types in the *Amazon EC2 User Guide*.

4. Choose **Select** to select the AMI.

5. Select an instance type, and then choose **Next: Configure Instance Details**.

6. **(For retired Amazon Linux AMI (AL1) platforms)** Skip this step if your environment runs on a supported Linux-based platform or on a Windows platform.

   Expand the **Advanced Details** section and paste the following text in the **User Data** field.

   ```
   #cloud-config
     repo_releasever: repository version number
     repo_upgrade: none
   ```

   The *repository version number* is the year and month version in the AMI name. For example, AMIs based on the March 2015 release of Amazon Linux have a repository version number `2015.03`. For an Elastic Beanstalk image, this matches the date shown in the solution stack name for your platform version based on Amazon Linux AMI (preceding Amazon Linux 2).

   > **Note**
   >
   > The `repo_releasever` setting configures the lock-on-launch feature for an Amazon Linux AMI. This causes the AMI to use a fixed, specific repository version when it launches. This feature isn't supported on Amazon Linux 2—don't specify it if your environment uses a current Amazon Linux 2 platform branch. The setting is required if you're using a custom AMI with Elastic Beanstalk only on Amazon Linux AMI platform branches (preceding Amazon Linux 2).
   > The `repo_upgrade` setting disables the automatic installation of security updates. It's required to use a custom AMI with Elastic Beanstalk.

7. Proceed through the wizard to launch the EC2 instance. When prompted, select a key pair that you have access to so that you can connect to the instance for the next steps.
8. Connect to the instance with SSH or RDP.
9. Perform any customizations you want.
10. **(Windows platforms)** Run the EC2Config service Sysprep. For information about EC2Config, see Configuring a Windows Instance Using the EC2Config Service. Ensure that Sysprep is configured to generate a random password that can be retrieved from the AWS Management Console.
11. In the Amazon EC2 console, stop the EC2 instance. Then on the **Instance Actions** menu, choose **Create Image (EBS AMI)**.
12. To avoid incurring additional AWS charges, terminate the EC2 instance.

To use your custom AMI in an Elastic Beanstalk environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. For **AMI ID**, enter your custom AMI ID.
6. To save the changes choose **Apply** at the bottom of the page.

When you create a new environment with the custom AMI, you should use the same platform version that you used as a base to create the AMI.

Managing an environment with a custom AMI

Platform updates

When using a custom AMI, Elastic Beanstalk will continue to use the same custom AMI in an environment when its platform version is updated, regardless of whether the update is applied manually or via managed platform updates. The environment will **not** be reset to use the stock AMI of the new platform version.

We recommend that you create a new custom AMI based on the stock AMI of the new platform version. Doing so will apply the patches available in the new platform version and will also minimize deployment failures due to incompatible package or library versions.

For more information about creating a new custom AMI, see the Creating a custom AMI earlier in this topic.

Removing a custom AMI

If you would like to remove a custom AMI from an environment and reset it to use the stock AMI for the environment's platform version, use the following CLI command.

```
aws elasticbeanstalk update-environment \
  --application-name my-application \
  --environment-name my-environment \
  --region us-east-1 \
  --options-to-remove Namespace=aws:autoscaling:launchconfiguration,OptionName=ImageId
```

Note

To avoid disruption of your service, test your application with a stock AMI before applying this change to your production environment.

Cleaning up a custom AMI

When you are done with a custom AMI and don't need it to launch Elastic Beanstalk environments anymore, consider cleaning it up to minimize storage cost. Cleaning up a custom AMI involves deregistering it from Amazon EC2 and deleting other associated resources. For details, see Deregistering Your Linux AMI or Deregistering Your Windows AMI.

Preserving access to an Amazon Machine Image (AMI) for a retired platform

Elastic Beanstalk sets a platform branch status to *retired* when the operating system or major component used by the branch reaches End of Life. The *base* Elastic Beanstalk AMI for the platform branch may also be made private to prevent the use of this out-of-date AMI. Environments using AMIs that have been made private will no longer be able to launch instances.

If you're unable to migrate your application to a supported environment before it's retired, your environment may be in this situation. The need to update an environment for a Beanstalk platform branch, where its base Elastic Beanstalk AMI has been made private, may arise. An alternative approach is available. You can update an existing environment based on a *copy* of the base Elastic Beanstalk AMI used by your environment.

This topic offers some steps and a standalone script to update an existing environment based on a *copy* of the base Elastic Beanstalk AMI used by your environment. Once you're able to migrate your application to a supported platform you can continue to use the standard procedures for maintaining your application and supported environments.

Manual steps

To update an environment based on an AMI copy of the base Elastic Beanstalk AMI

1. **Determine which AMI your environment is using.** This command returns the AMI used by the Elastic Beanstalk environment that you provide in the parameters. The returned value is used as the *source-ami-id* in the next step.

   In a command window, run a command like the following. For more information, see describe-configuration-settings in the *AWS CLI Command Reference*.

   Specify the AWS Region that stores the source AMI you want to copy. Replace the application name and environment name with those based on the source AMI. Enter the text for the query parameter as shown.

   Example

   ```
   >aws elasticbeanstalk describe-configuration-settings \
     --application-name my-application \
     --environment-name my-environment \
     --region us-east-2 \
     --query "ConfigurationSettings[0].OptionSettings[?OptionName=='ImageId'] |
    [0].Value"
   ```

2. **Copy the AMI into your account.** This command returns the new AMI that results from copying the *source-ami-id* that was returned in the prior step.

Note

Be sure to make a note of the new AMI id that is output by this command. You'll need to enter it in the next step, replacing *copied-ami-id* in the example command.

In a command window, run a command like the following. For more information, see copy-image in the *AWS CLI Command Reference*.

Specify the AWS Region of the source AMI you want to copy (**--source-region**) and the Region where you want to use your new custom AMI (**--region**). Replace *source-ami-id* with the AMI of the image that you're copying. The *source-ami-id* was returned by the command in the prior step. Replace *new-ami-name* with a name to describe the new AMI in the destination Region. The script that follows this procedure generates the new AMI name by appending the string "*Copy of*" to the beginning of the name of the *source-ami-id.*

```
>aws ec2 copy-image \
    --region us-east-2 \
    --source-image-id source-ami-id \
    --source-region us-east-2 \
    --name new-ami-name
```

3. **Update an environment to use the copied AMI.** After the command runs it returns the status of the environment.

In a command window, run a command like the following. For more information, see update-environment in the *AWS CLI Command Reference*.

Specify the AWS Region of the environment and application you need to update. Replace the application name and environment name with those you need to associate with the *copied-ami-id* from the prior step. For the **--option-setttings** parameter, replace `copied-ami-id` with the AMI id you noted from the output of the prior command.

```
>aws elasticbeanstalk update-environment \
  --application-name my-application \
  --environment-name my-environment \
  --region us-east-2 \
  --option-settings
 "Namespace=aws:autoscaling:launchconfiguration,OptionName=ImageId,Value=copied-
ami-id"
```

Note

To minimize storage costs, consider cleaning up your custom AMI when you don't need it to launch Elastic Beanstalk environments anymore. For more information, see Cleaning up a custom AMI.

Standalone script

The following script provides the same results as the previous manual steps. Download the script by selecting this link: copy_ami_and_update_env.zip.

Script source: copy_ami_and_update_env.sh

```
#!/bin/bash

set -ue

USAGE="This script is used to copy an AMI used by your Elastic Beanstalk environment
 into your account to use in your environment.\n\n"
USAGE+="Usage:\n\n"
USAGE+="./$(basename $0) [OPTIONS]\n"
USAGE+="OPTIONS:\n"
USAGE+="\t--application-name <application-name>\tThe name of your Elastic Beanstalk
 application.\n"
USAGE+="\t--environment-name <environment-name>\tThe name of your Elastic Beanstalk
 environment.\n"
USAGE+="\t--region <region> \t\t\tThe AWS region your Elastic Beanstalk environment is
 deployed to.\n"
USAGE+="\n\n"
USAGE+="Script Usage Example(s):\n"
USAGE+="./$(basename $0) --application-name my-application --environment-name my-
environment --region us-east-1\n"

if [ $# -eq 0 ]; then
  echo -e $USAGE
  exit
```

```
fi

while [[ $# -gt 0 ]]; do
  case $1 in
    --application-name)       APPLICATION_NAME="$2"; shift ;;
    --environment-name)       ENVIRONMENT_NAME="$2"; shift ;;
    --region)                 REGION="$2"; shift ;;
    *)                        echo "Unknown option $1" ; echo -e $USAGE ; exit ;;
  esac
  shift
done

aws_cli_version="$(aws --version)"
if [ $? -ne 0 ]; then
  echo "aws CLI not found. Please install it: https://docs.aws.amazon.com/cli/latest/
userguide/getting-started-install.html. Exiting."
  exit 1
fi
echo "Using aws CLI version: ${aws_cli_version}"

account=$(aws sts get-caller-identity --query "Account" --output text)
echo "Using account ${account}"

environment_ami_id=$(aws elasticbeanstalk describe-configuration-settings \
  --application-name "$APPLICATION_NAME" \
  --environment-name "$ENVIRONMENT_NAME" \
  --region "$REGION" \
  --query "ConfigurationSettings[0].OptionSettings[?OptionName=='ImageId'] | [0].Value"
 \
  --output text)
echo "Image associated with environment ${ENVIRONMENT_NAME} is ${environment_ami_id}"

owned_image=$(aws ec2 describe-images \
  --owners self \
  --image-ids "$environment_ami_id" \
  --region "$REGION" \
  --query "Images[0]" \
  --output text)
if [ "$owned_image" != "None" ]; then
  echo "${environment_ami_id} is already owned by account ${account}. Exiting."
  exit
fi

source_image_name=$(aws ec2 describe-images \
  --image-ids "$environment_ami_id" \
  --region "$REGION" \
  --query "Images[0].Name" \
  --output text)
if [ "$source_image_name" = "None" ]; then
  echo "Cannot find ${environment_ami_id}. Please contact AWS support if you need
 additional help: https://aws.amazon.com/support."
  exit 1
fi

copied_image_name="Copy of ${source_image_name}"
copied_ami_id=$(aws ec2 describe-images \
  --owners self \
  --filters Name=name,Values="${copied_image_name}" \
  --region "$REGION" \
  --query "Images[0].ImageId" \
  --output text)
if [ "$copied_ami_id" != "None" ]; then
  echo "Detected that ${environment_ami_id} has already been copied by account
 ${account}. Skipping image copy."
else
  echo "Copying ${environment_ami_id} to account ${account} with name
 ${copied_image_name}"
  copied_ami_id=$(aws ec2 copy-image \
    --source-image-id "$environment_ami_id" \
    --source-region "$REGION" \
    --name "$copied_image_name" \
    --region "$REGION" \
    --query "ImageId" \
    --output text)
  echo "New AMI ID is ${copied_ami_id}"

  echo "Waiting for ${copied_ami_id} to become available"
  aws ec2 wait image-available \
    --image-ids "$copied_ami_id" \
    --region "$REGION"
  echo "${copied_ami_id} is now available"
fi

echo "Updating environment ${ENVIRONMENT_NAME} to use ${copied_ami_id}"
environment_status=$(aws elasticbeanstalk update-environment \
  --application-name "$APPLICATION_NAME" \
  --environment-name "$ENVIRONMENT_NAME" \
  --option-settings
 "Namespace=aws:autoscaling:launchconfiguration,OptionName=ImageId,Value=
${copied_ami_id}" \
  --region "$REGION" \
  --query "Status" \
  --output text)
echo "Environment ${ENVIRONMENT_NAME} is now ${environment_status}"

echo "Waiting for environment ${ENVIRONMENT_NAME} update to complete"
aws elasticbeanstalk wait environment-updated \
  --application-name "$APPLICATION_NAME" \
  --environment-names "$ENVIRONMENT_NAME" \
  --region "$REGION"
echo "Environment ${ENVIRONMENT_NAME} update complete"
```

Note

You must have the AWS CLI installed to execute the script. For installation instructions, see Install or update the latest version of the AWS CLI in the *AWS Command Line Interface User Guide*.

After installing the AWS CLI, you must also configure it to use the AWS account that owns the environment. For more information, see Configure the AWS CLI in the *AWS Command Line Interface User Guide*. The account must also have permissions to create an AMI and update the Elastic Beanstalk environment.

These steps describe the process that the script follows.

1. Print the account in use.
2. Determine which AMI is used by the environment (source AMI).
3. Check if the source AMI is already owned by the account. If yes, exit.
4. Determine the name of the source AMI so it can be used in the new AMI name. This also serves to confirm access to the source AMI.
5. Check if the source AMI has already been copied to the account. This is done by searching for AMIs with the name of the copied AMI owned by the account. If the AMI name has been changed in between script executions, it will copy the image again.
6. If the source AMI has not already been copied, copy the source AMI to the account and wait for the new AMI to be available.

7. Update the environment configuration to use the new AMI.
8. Wait for the environment update to complete.

After you extract the script from the copy_ami_and_update_env.zip file, run it by executing the following example. Replace the application name and environment name in the example with your own values.

```
>sh copy_ami_and_update_env.sh \
  --application-name my-application \
  --environment-name my-environment \
  --region us-east-1
```

Note

To minimize storage costs, consider cleaning up your custom AMI when you don't need it to launch Elastic Beanstalk environments anymore. For more information, see Cleaning up a custom AMI.

Serving static files

To improve performance, you can configure the proxy server to serve static files (for example, HTML or images) from a set of directories inside your web application. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application.

Elastic Beanstalk supports configuring the proxy to serve static files on most platform branches based on Amazon Linux 2. The one exception is Docker.

Note

On the Python and Ruby platforms, Elastic Beanstalk configures some static file folders by default. For details, see the static file configuration sections for Python and Ruby. You can configure additional folders as explained on this page.

Configure static files using the console

To configure the proxy server to serve static files

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. Scroll to the **Platform software** section and locate the **Static files** group.

   a. To add a static file mapping, select **Add static files**. In the extra row that appears you'll enter a *path* for serving static files and the *directory* that contains the static files to serve.

      - In the **Path** field, start the path name with a slash (/) (for example, "*/images*").
      - In the **Directory** field, specify a directory name located in the root of your application's source code. Don't start it with a slash (for example, "*static/image-files*").

      Note

      If you aren't seeing the **Static files** section, you have to add at least one mapping by using a configuration file. For details, see the section called "Configure static files using configuration options" on this page.

   b. To remove a mapping, select **Remove**.
6. To save the changes choose **Apply** at the bottom of the page.

Configure static files using configuration options

You can use a configuration file to configure static file paths and directory locations using configuration options. You can add a configuration file to your application's source bundle and deploy it during environment creation or a later deployment.

If your environment uses a platform branch based on Amazon Linux 2, use the `aws:elasticbeanstalk:environment:proxy:staticfiles` namespace.

The following example configuration file tells the proxy server to serve files in the `statichtml` folder at the path `/html`, and files in the `staticimages` folder at the path `/images`.

Example .ebextensions/static-files.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:staticfiles:
    /html: statichtml
    /images: staticimages
```

If your Elastic Beanstalk environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the following additional information:

Amazon Linux AMI platform-specific namespaces

On Amazon Linux AMI platform branches, static file configuration namespaces vary by platform. For details, see one of the following pages:

- Go configuration namespace
- Java SE configuration namespace
- Tomcat configuration namespaces
- Node.js configuration namespace
- Python configuration namespaces

Configuring HTTPS for your Elastic Beanstalk environment

This topics in this section explain how to configure HTTPS for your Elastic Beanstalk environment. HTTPS is a must for any application that transmits user data or login information.

If you've purchased and configured a custom domain name for your Elastic Beanstalk environment, you can use HTTPS to allow users to connect to your web site securely.

If you don't own a domain name, you can still use HTTPS with a self-signed certificate for development and testing purposes. For more information, see Server certificates.

Configuring HTTPS Termination at the load balancer

A load balancer distributes requests to the EC2 instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet. The simplest way to use HTTPS with an Elastic Beanstalk multi-instance environment is to configure a secure listener for the load balancer. The connection between the client and the load balancer remains secure, so you can configure the load balancer to terminate HTTPS. The back end connections between the load balancer and EC2 instances use HTTP, so no additional configuration of the instances is required. For detailed instructions to configure a secure listenter, see Configuring HTTPS Termination at the load balancer.

Configuring HTTPS Termination at the EC2 instance

If you run your application in a single instance environment, or need to secure the connection all the way to the EC2 instances behind the load balancer, you can configure the proxy server that runs on the instance to terminate HTTPS. Configuring your instances to terminate HTTPS connections requires the use of configuration files to modify the software running on the instances, and to modify security groups to allow secure connections. For more information, see Configuring HTTPS Termination at the instance.

Configuring HTTPS end-to-end

For end-to-end HTTPS in a load-balanced environment, you can combine instance and load balancer termination to encrypt both connections. By default, if you configure the load balancer to forward traffic using HTTPS, it will trust any certificate presented to it by the backend instances. For maximum security, you can attach policies to the load balancer that prevent it from connecting to instances that don't present a public certificate that it trusts. For more information, see Configuring end-to-end encryption in a load-balanced Elastic Beanstalk environment.

Configuring HTTPS with TCP Passthrough

You can also configure the load balancer to relay HTTPS traffic without decrypting it. For more information, see Configuring your environment's load balancer for TCP Passthrough.

Note

The Does it have Snakes? sample application on GitHub includes configuration files and instructions for each method of configuring HTTPS with a Tomcat web application. See the readme file and HTTPS instructions for details.

Topics

- Configuring HTTPS Termination at the instance
- Configuring end-to-end encryption in a load-balanced Elastic Beanstalk environment
- Configuring your environment's load balancer for TCP Passthrough
- Configuring HTTP to HTTPS redirection

Server certificates

This topic describes the different types of certificates you can use to configure HTTPS and when to apply each. The subtopics in this section provide instructions to create your own certificate and how to upload it.

AWS Certificate Manager (ACM)

ACM is the preferred tool to provision, manage, and deploy your server certificates. You can do so programmatically or using the AWS CLI. With ACM you can create a trusted certificate for your domain names for free.

ACM certificates can only be used with AWS load balancers and Amazon CloudFront distributions, and ACM is available only in certain AWS Regions. To use an ACM certificate with Elastic Beanstalk, see Configuring HTTPS Termination at the load balancer. For more information about ACM see the *AWS Certificate Manager User Guide*.

Note

For a list of regions where ACM is available, see ACM endpoints and quotas in the *Amazon Web Services General Reference*.

If ACM is not available in your AWS Region, you can upload a third-party or self-signed certificate and private key to AWS Identity and Access Management (IAM). You can use the AWS CLI to upload the certificate. Certificates stored in IAM can be used with load balancers and CloudFront distributions. For more information, see Upload a certificate to IAM.

Third party certificate

If ACM is not available in your region, you can purchase a trusted certificate from a third party. A third-party certificate can be used to decrypt HTTPS traffic at your load balancer, on the backend instances, or both.

Self-signed certificate

For development and testing, you can create and sign a certificate yourself with open source tools. Self-signed certificates are free and easy to create, but cannot be used for front-end decryption on public sites. If you attempt to use a self-signed certificate for an HTTPS connection to a client, the user's browser displays an error message indicating that your web site is unsafe. You can, however, use a self-signed certificate to secure backend connections without issue.

Create and sign an X509 certificate

You can create an X509 certificate for your application with `OpenSSL`. OpenSSL is a standard, open source library that supports a wide range of cryptographic functions, including the creation and signing of x509 certificates. For more information about OpenSSL, visit www.openssl.org.

Note

You only need to create a certificate locally if you want to use HTTPS in a single instance environment or re-encrypt on the backend with a self-signed certificate. If you own a domain name, you can create a certificate in AWS and use it with a load-balanced environment for free by using AWS Certificate Manager (ACM). See Request a Certificate in the *AWS Certificate Manager User Guide* for instructions.

Run `openssl version` at the command line to see if you already have OpenSSL installed. If you don't, you can build and install the source code using the instructions at the public GitHub repository, or use your favorite package manager. OpenSSL is also installed on Elastic Beanstalk's Linux images, so a quick alternative is to connect to an EC2 instance in a running environment by using the EB CLI's **eb ssh** command:

```
~/eb$ eb ssh
[ec2-user@ip-255-55-55-255 ~]$ openssl version
OpenSSL 1.0.1k-fips 8 Jan 2015
```

You need to create an RSA private key to create your certificate signing request (CSR). To create your private key, use the **openssl genrsa** command:

```
[ec2-user@ip-255-55-55-255 ~]$ openssl genrsa 2048 > privatekey.pem
Generating RSA private key, 2048 bit long modulus
.............................................................................................
+++
...............+++
e is 65537 (0x10001)
```

privatekey.pem

The name of the file where you want to save the private key. Normally, the **openssl genrsa** command prints the private key contents to the screen, but this command pipes the output to a file. Choose any file name, and store the file in a secure place so that you can retrieve it later. If you lose your private key, you won't be able to use your certificate.

A CSR is a file you send to a certificate authority (CA) to apply for a digital server certificate. To create a CSR, use the **openssl req** command:

```
$ openssl req -new -key privatekey.pem -out csr.pem
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
```

Enter the information requested and press **Enter**. The following table describes and shows examples for each field.

Name	Description	Example
Country Name	The two-letter ISO abbreviation for your country.	US = United States
State or Province	The name of the state or province where your organization is located. You cannot abbreviate this name.	Washington
Locality Name	The name of the city where your organizat ion is located.	Seattle
Organization Name	The full legal name of your organization. Do not abbreviate your organization name.	Example Corporation

Name	Description	Example
Organizational Unit	Optional, for additional organization information.	Marketing
Common Name	The fully qualified domain name for your web site. This must match the domain name that users see when they visit your site, otherwise certificate errors will be shown.	www.example.com
Email address	The site administrator's email address.	someone@example.com

You can submit the signing request to a third party for signing, or sign it yourself for development and testing. Self-signed certificates can also be used for backend HTTPS between a load balancer and EC2 instances.

To sign the certificate, use the **openssl x509** command. The following example uses the private key from the previous step (*privatekey.pem*) and the signing request (*csr.pem*) to create a public certificate named *public.crt* that is valid for *365* days.

```
$ openssl x509 -req -days 365 -in csr.pem -signkey privatekey.pem -out public.crt
Signature ok
subject=/C=us/ST=washington/L=seattle/O=example corporation/OU=marketing/
CN=www.example.com/emailAddress=someone@example.com
Getting Private key
```

Keep the private key and public certificate for later use. You can discard the signing request. Always store the private key in a secure location and avoid adding it to your source code.

To use the certificate with the Windows Server platform, you must convert it to a PFX format. Use the following command to create a PFX certificate from the private key and public certificate files:

```
$ openssl pkcs12 -export -out example.com.pfx -inkey privatekey.pem -in public.crt
Enter Export Password: password
Verifying - Enter Export Password: password
```

Now that you have a certificate, you can upload it to IAM for use with a load balancer, or configure the instances in your environment to terminate HTTPS.

Upload a certificate to IAM

To use your certificate with your Elastic Beanstalk environment's load balancer, upload the certificate and private key to AWS Identity and Access Management (IAM). You can use a certificate stored in IAM with Elastic Load Balancing load balancers and Amazon CloudFront distributions.

Note

AWS Certificate Manager (ACM) is the preferred tool to provision, manage, and deploy your server certificates. For more information about requesting an ACM certificate, see Request a Certificate in the *AWS Certificate Manager User Guide*. For more information about importing third-party certificates into ACM, see Importing Certificates in the *AWS Certificate Manager User Guide*. Use IAM to upload a certificate only if ACM is not available in your AWS Region.

You can use the AWS Command Line Interface (AWS CLI) to upload your certificate. The following command uploads a self-signed certificate named *https-cert.crt* with a private key named *private-key.pem*:

```
$ aws iam upload-server-certificate --server-certificate-name elastic-beanstalk-x509 --
certificate-body file://https-cert.crt --private-key file://private-key.pem
{
    "ServerCertificateMetadata": {
        "ServerCertificateId": "AS5YBEIONO2Q7CAIHKNGC",
        "ServerCertificateName": "elastic-beanstalk-x509",
        "Expiration": "2017-01-31T23:06:22Z",
        "Path": "/",
        "Arn": "arn:aws:iam::123456789012:server-certificate/elastic-beanstalk-x509",
        "UploadDate": "2016-02-01T23:10:34.167Z"
    }
}
```

The `file://` prefix tells the AWS CLI to load the contents of a file in the current directory. *elastic-beanstalk-x509* specifies the name to call the certificate in IAM.

If you purchased a certificate from a certificate authority and received a certificate chain file, upload that as well by including the `--certificate-chain` option:

```
$ aws iam upload-server-certificate --server-certificate-name elastic-beanstalk-x509 --
certificate-chain file://certificate-chain.pem --certificate-body file://https-cert.crt
 --private-key file://private-key.pem
```

Make note of the Amazon Resource Name (ARN) for your certificate. You'll use it when you update your load balancer configuration settings to use HTTPS.

Note

A certificate uploaded to IAM stays stored even after it's no longer used in any environment's load balancer. It contains sensitive data. When you no longer need the certificate for any environment, be sure to delete it. For details about deleting a certificate from IAM, see https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_server-certs.html#delete-server-certificate.

For more information about server certificates in IAM, see Working with Server Certificates in the *IAM User Guide*.

Storing private keys securely in Amazon S3

The private key that you use to sign your public certificate is private and should not be committed to source code. You can avoid storing private keys in configuration files by uploading them to Amazon S3, and configuring Elastic Beanstalk to download the file from Amazon S3 during application deployment.

The following example shows the Resources and files sections of a configuration file downloads a private key file from an Amazon S3 bucket.

Example .ebextensions/privatekey.config

```
Resources:
  AWSEBAutoScalingGroup:
    Metadata:
      AWS::CloudFormation::Authentication:
        S3Auth:
          type: "s3"
          buckets: ["elasticbeanstalk-us-west-2-123456789012"]
          roleName:
            "Fn::GetOptionSetting":
              Namespace: "aws:autoscaling:launchconfiguration"
              OptionName: "IamInstanceProfile"
              DefaultValue: "aws-elasticbeanstalk-ec2-role"
files:
  # Private key
  "/etc/pki/tls/certs/server.key":
    mode: "000400"
    owner: root
    group: root
    authentication: "S3Auth"
    source: https://elasticbeanstalk-us-west-2-123456789012.s3.us-west-2.amazonaws.com/
server.key
```

Replace the bucket name and URL in the example with your own. The first entry in this file adds an authentication method named `S3Auth` to the environment's Auto Scaling group's metadata. If you have configured a custom instance profile for your environment, that will be used, otherwise the default value of `aws-elasticbeanstalk-ec2-role` is applied. The default instance profile has permission to read from the Elastic Beanstalk storage bucket. If you use a different bucket, add permissions to the instance profile.

The second entry uses the `S3Auth` authentication method to download the private key from the specified URL and save it to `/etc/pki/tls/certs/server.key`. The proxy server can then read the private key from this location to terminate HTTPS connections at the instance.

The instance profile assigned to your environment's EC2 instances must have permission to read the key object from the specified bucket. Verify that the instance profile has permission to read the object in IAM, and that the permissions on the bucket and object do not prohibit the instance profile.

To view a bucket's permissions

1. Open the Amazon S3 Management Console.
2. Choose a bucket.
3. Choose **Properties** and then choose **Permissions**.
4. Verify that your account is a grantee on the bucket with read permission.
5. If a bucket policy is attached, choose **Bucket policy** to view the permissions assigned to the bucket.

Configuring HTTPS Termination at the load balancer

To update your AWS Elastic Beanstalk environment to use HTTPS, you need to configure an HTTPS listener for the load balancer in your environment. Two types of load balancer support an HTTPS listener: Classic Load Balancer and Application Load Balancer.

You can use the Elastic Beanstalk console or a configuration file to configure a secure listener and assign the certificate.

> **Note**
>
> Single-instance environments don't have a load balancer and don't support HTTPS termination at the load balancer.

Configuring a secure listener using the Elastic Beanstalk console

To assign a certificate to your environment's load balancer

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Load balancer** configuration category, choose **Edit**.

   > **Note**
   >
   > If the **Load balancer** configuration category doesn't have an **Edit** button, your environment doesn't have a load balancer.

5. On the **Modify load balancer** page, the procedure varies depending on the type of load balancer associated with your environment.
   - **Classic Load Balancer**
     a. Choose **Add listener**.
     b. In the **Classic Load Balancer listener** dialog box, configure the following settings:
        - For **Listener port**, type the incoming traffic port, typically 443.

    - For **Listener protocol**, choose **HTTPS**.
    - For **Instance port**, type `80`.
    - For **Instance protocol**, choose **HTTP**.
    - For **SSL certificate**, choose your certificate.

  c. Choose **Add**.

- **Application Load Balancer**

  a. Choose **Add listener**.

  b. In the **Application Load Balancer listener** dialog box, configure the following settings:

    - For **Port**, type the incoming traffic port, typically `443`.
    - For **Protocol**, choose **HTTPS**.
    - For **SSL certificate**, choose your certificate.

  c. Choose **Add**.

  Note

  For Classic Load Balancer and Application Load Balancer, if the drop-down menu doesn't show any certificates, you should create or upload a certificate for your custom domain name in AWS Certificate Manager (ACM) (preferred). Alternatively, upload a certificate to IAM with the AWS CLI.

- **Network Load Balancer**

  a. Choose **Add listener**.

  b. In the **Network Load Balancer listener** dialog box, for **Port**, type the incoming traffic port, typically `443`.

  c. Choose **Add**.

6. To save the changes choose **Apply** at the bottom of the page.

Configuring a secure listener using a configuration file

You can configure a secure listener on your load balancer with one of the following configuration files.

Example .ebextensions/securelistener-clb.config

Use this example when your environment has a Classic Load Balancer. The example uses options in the `aws:elb:listener` namespace to configure an HTTPS listener on port 443 with the specified certificate, and to forward the decrypted traffic to the instances in your environment on port 80.

```
option_settings:
  aws:elb:listener:443:
    SSLCertificateId: arn:aws:acm:us-east-2:1234567890123:certificate/
####################################
    ListenerProtocol: HTTPS
    InstancePort: 80
```

Replace the highlighted text with the ARN of your certificate. The certificate can be one that you created or uploaded in AWS Certificate Manager (ACM) (preferred), or one that you uploaded to IAM with the AWS CLI.

For more information about Classic Load Balancer configuration options, see Classic Load Balancer configuration namespaces.

Example .ebextensions/securelistener-alb.config

Use this example when your environment has an Application Load Balancer. The example uses options in the `aws:elbv2:listener` namespace to configure an HTTPS listener on port 443 with the specified certificate. The listener routes traffic to the default process.

```
option_settings:
  aws:elbv2:listener:443:
    ListenerEnabled: 'true'
    Protocol: HTTPS
    SSLCertificateArns: arn:aws:acm:us-east-2:1234567890123:certificate/
####################################
```

Example .ebextensions/securelistener-nlb.config

Use this example when your environment has a Network Load Balancer. The example uses options in the `aws:elbv2:listener` namespace to configure a listener on port 443. The listener routes traffic to the default process.

```
option_settings:
  aws:elbv2:listener:443:
    ListenerEnabled: 'true'
```

Configuring a security group

If you configure your load balancer to forward traffic to an instance port other than port 80, you must add a rule to your security group that allows inbound traffic over the instance port from your load balancer. If you create your environment in a custom VPC, Elastic Beanstalk adds this rule for you.

You add this rule by adding a `Resources` key to a configuration file in the `.ebextensions` directory for your application.

The following example configuration file adds an ingress rule to the `AWSEBSecurityGroup` security group. This allows traffic on port 1000 from the load balancer's security group.

Example .ebextensions/sg-ingressfromlb.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 1000
      FromPort: 1000
      SourceSecurityGroupId: {"Fn::GetAtt" : ["AWSEBLoadBalancerSecurityGroup", 
 "GroupId"]}
```

Configuring HTTPS Termination at the instance

You can use configuration files to configure the proxy server that passes traffic to your application to terminate HTTPS connections. This is useful if you want to use HTTPS with a single instance environment, or if you configure your load balancer to pass traffic through without decrypting it.

To enable HTTPS, you must allow incoming traffic on port 443 to the EC2 instance that your Elastic Beanstalk application is running on. You do this by using the `Resources` key in the configuration file to add a rule for port 443 to the ingress rules for the AWSEBSecurityGroup security group.

The following snippet adds an ingress rule to the `AWSEBSecurityGroup` security group that opens port 443 to all traffic for a single instance environment:

.ebextensions/https-instance-securitygroup.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

In a load-balanced environment in a default Amazon Virtual Private Cloud (Amazon VPC), you can modify this policy to only accept traffic from the load balancer. See Configuring end-to-end encryption in a load-balanced Elastic Beanstalk environment for an example.

Platforms

- Terminating HTTPS on EC2 instances running Docker
- Terminating HTTPS on EC2 instances running Go
- Terminating HTTPS on EC2 instances running Java SE
- Terminating HTTPS on EC2 instances running Node.js
- Terminating HTTPS on EC2 instances running PHP
- Terminating HTTPS on EC2 instances running Python
- Terminating HTTPS on EC2 instances running Ruby
- Terminating HTTPS on EC2 instances running Tomcat
- Terminating HTTPS on Amazon EC2 instances running .NET Core on Linux
- Terminating HTTPS on Amazon EC2 instances running .NET

Terminating HTTPS on EC2 instances running Docker

For Docker containers, you use a configuration file to enable HTTPS.

Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `files` key creates the following files on the instance:

`/etc/nginx/conf.d/https.conf`

Configures the nginx server. This file is loaded when the nginx service starts.

`/etc/pki/tls/certs/server.crt`

Creates the certificate file on the instance. Replace *`certificate file contents`* with the contents of your certificate.

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

If you have intermediate certificates, include them in `server.crt` after your site certificate.

```
    -----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

`/etc/pki/tls/certs/server.key`

Creates the private key file on the instance. Replace *`private key contents`* with the contents of the private key used to create the certificate request or self-signed certificate.

Example .ebextensions/https-instance.config

```
files:
  /etc/nginx/conf.d/https.conf:
    mode: "000644"
    owner: root
    group: root
    content: |
      # HTTPS Server

      server {
        listen 443;
        server_name localhost;

        ssl on;
        ssl_certificate /etc/pki/tls/certs/server.crt;
        ssl_certificate_key /etc/pki/tls/certs/server.key;

        ssl_session_timeout 5m;

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        ssl_prefer_server_ciphers on;

        location / {
          proxy_pass http://docker;
          proxy_http_version 1.1;

          proxy_set_header Connection "";
          proxy_set_header Host $host;
          proxy_set_header X-Real-IP $remote_addr;
          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
          proxy_set_header X-Forwarded-Proto https;
        }
      }

  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
    -----END RSA PRIVATE KEY-----
```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see Storing private keys securely in Amazon S3.

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an AWS CloudFormation function and adds a rule to it.

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load-balanced environment, you configure the load balancer to either pass secure traffic through untouched, or decrypt and re-encrypt for end-to-end encryption.

Terminating HTTPS on EC2 instances running Go

For Go container types, you enable HTTPS with a configuration file and an nginx configuration file that configures the nginx server to use HTTPS.

Add the following snippet to your configuration file, replacing the certificate and private key placeholders as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `Resources` key enables port 443 on the security group used by your environment's instance.

- The `files` key creates the following files on the instance:

  `/etc/pki/tls/certs/server.crt`

  Creates the certificate file on the instance. Replace *`certificate file contents`* with the contents of your certificate.

  > **Note**
  >
  > YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

  If you have intermediate certificates, include them in `server.crt` after your site certificate.

  ```
      -----BEGIN CERTIFICATE-----
  certificate file contents
  -----END CERTIFICATE-----
  -----BEGIN CERTIFICATE-----
  first intermediate certificate
  -----END CERTIFICATE-----
  -----BEGIN CERTIFICATE-----
  second intermediate certificate
  -----END CERTIFICATE-----
  ```

  `/etc/pki/tls/certs/server.key`

  Creates the private key file on the instance. Replace *`private key contents`* with the contents of the private key used to create the certificate request or self-signed certificate.

- The `container_commands` key restarts the nginx server after everything is configured so that the server loads the nginx configuration file.

Example .ebextensions/https-instance.config

```
files:
  /etc/pki/tls/certs/server.crt:
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
```

```
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----

container_commands:
  01restart_nginx:
    command: "service nginx restart"
```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see Storing private keys securely in Amazon S3.

Place the following in a file with the `.conf` extension in the `.ebextensions/nginx/conf.d/` directory of your source bundle (e.g., `.ebextensions/nginx/conf.d/https.conf`). Replace *`app_port`* with the port number that your application listens on. This example configures the nginx server to listen on port 443 using SSL. For more information about these configuration files on the Go platform, see Configuring the proxy server.

Example .ebextensions/nginx/conf.d/https.conf

```
# HTTPS server

server {
    listen       443;
    server_name  localhost;

    ssl                  on;
    ssl_certificate      /etc/pki/tls/certs/server.crt;
    ssl_certificate_key  /etc/pki/tls/certs/server.key;

    ssl_session_timeout  5m;

    ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers   on;

    location / {
        proxy_pass  http://localhost:app_port;
        proxy_set_header   Connection "";
        proxy_http_version 1.1;
        proxy_set_header        Host            $host;
        proxy_set_header        X-Real-IP       $remote_addr;
        proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header        X-Forwarded-Proto https;
    }
}
```

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an AWS CloudFormation function and adds a rule to it.

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load-balanced environment, you configure the load balancer to either pass secure traffic through untouched, or decrypt and re-encrypt for end-to-end encryption.

Terminating HTTPS on EC2 instances running Java SE

For Java SE container types, you enable HTTPS with an .ebextensions configuration file, and an nginx configuration file that configures the nginx server to use HTTPS.

All AL2023/AL2 platforms support a uniform proxy configuration feature. For more information about configuring the proxy server on your platform versions running AL2023/AL2, see Reverse proxy configuration.

Add the following snippet to your configuration file, replacing the certificate and private key placeholders as instructed, and save it in the `.ebextensions` directory. The configuration file performs the following tasks:

- The `files` key creates the following files on the instance:

  `/etc/pki/tls/certs/server.crt`

  Creates the certificate file on the instance. Replace *`certificate file contents`* with the contents of your certificate.

  Note

  YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

  If you have intermediate certificates, include them in `server.crt` after your site certificate.

  ```
      -----BEGIN CERTIFICATE-----
  certificate file contents
  -----END CERTIFICATE-----
  -----BEGIN CERTIFICATE-----
  first intermediate certificate
  -----END CERTIFICATE-----
  -----BEGIN CERTIFICATE-----
  second intermediate certificate
  -----END CERTIFICATE-----
  ```

  `/etc/pki/tls/certs/server.key`

  Creates the private key file on the instance. Replace *`private key contents`* with the contents of the private key used to create the certificate request or self-signed certificate.

- The `container_commands` key restarts the nginx server after everything is configured so that the server loads the nginx configuration file.

Example .ebextensions/https-instance.config

```
files:
```

```
  /etc/pki/tls/certs/server.crt:
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----

container_commands:
  01restart_nginx:
    command: "service nginx restart"
```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see Storing private keys securely in Amazon S3.

Place the following in a file with the `.conf` extension in the `.ebextensions/nginx/conf.d/` directory of your source bundle (e.g., `.ebextensions/nginx/conf.d/https.conf`). Replace *`app_port`* with the port number that your application listens on. This example configures the nginx server to listen on port 443 using SSL. For more information about these configuration files on the Java SE platform, see Configuring the proxy server.

Example .ebextensions/nginx/conf.d/https.conf

```
# HTTPS server

server {
    listen       443;
    server_name  localhost;

    ssl                  on;
    ssl_certificate      /etc/pki/tls/certs/server.crt;
    ssl_certificate_key  /etc/pki/tls/certs/server.key;
    ssl_session_timeout  5m;

    ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers   on;

    location / {
        proxy_pass  http://localhost:app_port;
        proxy_set_header   Connection "";
        proxy_http_version 1.1;
        proxy_set_header        Host            $host;
        proxy_set_header        X-Real-IP       $remote_addr;
        proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header        X-Forwarded-Proto https;
    }
}
```

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an AWS CloudFormation function and adds a rule to it.

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load-balanced environment, you configure the load balancer to either pass secure traffic through untouched, or decrypt and re-encrypt for end-to-end encryption.

Terminating HTTPS on EC2 instances running Node.js

The following example configuration file extends the default nginx configuration to listen on port 443 and terminate SSL/TLS connections with a public certificate and private key.

If you configured your environment for enhanced health reporting, you need to configure nginx to generate access logs. To do that, uncomment the block of lines under the comment that reads `# For enhanced health...` by removing the leading # characters.

Example .ebextensions/https-instance.config

```
files:
  /etc/nginx/conf.d/https.conf:
    mode: "000644"
    owner: root
    group: root
    content: |
      # HTTPS server

      server {
          listen       443;
          server_name  localhost;

          ssl                  on;
          ssl_certificate      /etc/pki/tls/certs/server.crt;
          ssl_certificate_key  /etc/pki/tls/certs/server.key;

          ssl_session_timeout  5m;

          ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
          ssl_prefer_server_ciphers   on;

          # For enhanced health reporting support, uncomment this block:

          #if ($time_iso8601 ~ "^(\d{4})-(\d{2})-(\d{2})T(\d{2})") {
          #    set $year $1;
          #    set $month $2;
          #    set $day $3;
          #    set $hour $4;
          #}
          #access_log /var/log/nginx/healthd/application.log.$year-$month-$day-$hour
 healthd;
          #access_log  /var/log/nginx/access.log  main;

          location / {
              proxy_pass  http://nodejs;
              proxy_set_header   Connection "";
              proxy_http_version 1.1;
              proxy_set_header        Host            $host;
              proxy_set_header        X-Real-IP       $remote_addr;
              proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
              proxy_set_header        X-Forwarded-Proto https;
          }
      }

  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----
```

The `files` key creates the following files on the instance:

`/etc/nginx/conf.d/https.conf`

Configures the nginx server. This file is loaded when the nginx service starts.

`/etc/pki/tls/certs/server.crt`

Creates the certificate file on the instance. Replace *`certificate file contents`* with the contents of your certificate.

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

If you have intermediate certificates, include them in `server.crt` after your site certificate.

```
    -----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

`/etc/pki/tls/certs/server.key`

Creates the private key file on the instance. Replace *`private key contents`* with the contents of the private key used to create the certificate request or self-signed certificate.

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see Storing private keys securely in Amazon S3.

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an AWS CloudFormation function and adds a rule to it.

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load-balanced environment, you configure the load balancer to either pass secure traffic through untouched, or decrypt and re-encrypt for end-to-end encryption.

Terminating HTTPS on EC2 instances running PHP

For PHP container types, you use a configuration file to enable the Apache HTTP Server to use HTTPS.

Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory.

The configuration file performs the following tasks:

- The `packages` key uses yum to install `mod24_ssl`.
- The `files` key creates the following files on the instance:

  `/etc/httpd/conf.d/ssl.conf`

    Configures the Apache server. This file loads when the Apache service starts.

  `/etc/pki/tls/certs/server.crt`

    Creates the certificate file on the instance. Replace *`certificate file contents`* with the contents of your certificate.

    Note

    YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

    If you have intermediate certificates, include them in `server.crt` after your site certificate.

```
    -----BEGIN CERTIFICATE-----
  certificate file contents
  -----END CERTIFICATE-----
  -----BEGIN CERTIFICATE-----
  first intermediate certificate
  -----END CERTIFICATE-----
  -----BEGIN CERTIFICATE-----
  second intermediate certificate
```

```
-----END CERTIFICATE-----
```

`/etc/pki/tls/certs/server.key`

Creates the private key file on the instance. Replace *`private key contents`* with the contents of the private key used to create the certificate request or self-signed certificate.

Example .ebextensions/https-instance.config

```
packages:
  yum:
    mod24_ssl : []

files:
  /etc/httpd/conf.d/ssl.conf:
    mode: "000644"
    owner: root
    group: root
    content: |
      LoadModule ssl_module modules/mod_ssl.so
      Listen 443
      <VirtualHost *:443>
        <Proxy *>
          Order deny,allow
          Allow from all
        </Proxy>

        SSLEngine               on
        SSLCertificateFile      "/etc/pki/tls/certs/server.crt"
        SSLCertificateKeyFile   "/etc/pki/tls/certs/server.key"
        SSLCipherSuite          EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
        SSLProtocol             All -SSLv2 -SSLv3
        SSLHonorCipherOrder     On
        SSLSessionTickets       Off

        Header always set Strict-Transport-Security "max-age=63072000;
 includeSubdomains; preload"
        Header always set X-Frame-Options DENY
        Header always set X-Content-Type-Options nosniff

        ProxyPass / http://localhost:80/ retry=0
        ProxyPassReverse / http://localhost:80/
        ProxyPreserveHost on
        RequestHeader set X-Forwarded-Proto "https" early

      </VirtualHost>

  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----
```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see Storing private keys securely in Amazon S3.

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an AWS CloudFormation function and adds a rule to it.

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
```

```
        IpProtocol: tcp
        ToPort: 443
        FromPort: 443
        CidrIp: 0.0.0.0/0
```

For a load-balanced environment, you configure the load balancer to either pass secure traffic through untouched, or decrypt and re-encrypt for end-to-end encryption.

Terminating HTTPS on EC2 instances running Python

For Python container types using Apache HTTP Server with the Web Server Gateway Interface (WSGI), you use a configuration file to enable the Apache HTTP Server to use HTTPS.

Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `packages` key uses yum to install `mod24_ssl`.
- The `files` key creates the following files on the instance:

  `/etc/httpd/conf.d/ssl.conf`

  Configures the Apache server. If your application is not named `application.py`, replace the highlighted text in the value for `WSGIScriptAlias` with the local path to your application. For example, a django application's may be at `django/wsgi.py`. The location should match the value of the `WSGIPath` option that you set for your environment.

  Depending on your application requirements, you may also need to add other directories to the **python-path** parameter.

  `/etc/pki/tls/certs/server.crt`

  Creates the certificate file on the instance. Replace *`certificate file contents`* with the contents of your certificate.

  Note

  YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

If you have intermediate certificates, include them in `server.crt` after your site certificate.

```
    -----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

`/etc/pki/tls/certs/server.key`

Creates the private key file on the instance. Replace *`private key contents`* with the contents of the private key used to create the certificate request or self-signed certificate.

- The `container_commands` key stops the httpd service after everything has been configured so that the service uses the new `https.conf` file and certificate.

Note

The example works only in environments using the Python platform.

Example .ebextensions/https-instance.config

```
packages:
  yum:
    mod24_ssl : []

files:
  /etc/httpd/conf.d/ssl.conf:
    mode: "000644"
    owner: root
    group: root
    content: |
      LoadModule wsgi_module modules/mod_wsgi.so
      WSGIPythonHome /opt/python/run/baselinenv
      WSGISocketPrefix run/wsgi
      WSGIRestrictEmbedded On
      Listen 443
      <VirtualHost *:443>
        SSLEngine on
        SSLCertificateFile "/etc/pki/tls/certs/server.crt"
        SSLCertificateKeyFile "/etc/pki/tls/certs/server.key"

        Alias /static/ /opt/python/current/app/static/
        <Directory /opt/python/current/app/static>
        Order allow,deny
        Allow from all
        </Directory>

        WSGIScriptAlias / /opt/python/current/app/application.py

        <Directory /opt/python/current/app>
        Require all granted
        </Directory>

        WSGIDaemonProcess wsgi-ssl processes=1 threads=15 display-name=%{GROUP} \
          python-path=/opt/python/current/app \
          python-home=/opt/python/run/venv \
          home=/opt/python/current/app \
          user=wsgi \
          group=wsgi
        WSGIProcessGroup wsgi-ssl

      </VirtualHost>

  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----

container_commands:
  01killhttpd:
    command: "killall httpd"
  02waitforhttpddeath:
    command: "sleep 3"
```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see Storing private keys securely in Amazon S3.

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an AWS CloudFormation function and adds a rule to it.

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load-balanced environment, you configure the load balancer to either pass secure traffic through untouched, or decrypt and re-encrypt for end-to-end encryption.

Terminating HTTPS on EC2 instances running Ruby

For Ruby container types, the way you enable HTTPS depends on the type of application server used.

Topics

Configure HTTPS for Ruby with Puma

For Ruby container types that use Puma as the application server, you use a configuration file to enable HTTPS.

Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `files` key creates the following files on the instance:

  `/etc/nginx/conf.d/https.conf`

  Configures the nginx server. This file is loaded when the nginx service starts.

  `/etc/pki/tls/certs/server.crt`

  Creates the certificate file on the instance. Replace *`certificate file contents`* with the contents of your certificate.

  Note

  YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

  If you have intermediate certificates, include them in `server.crt` after your site certificate.

```
    -----BEGIN CERTIFICATE-----
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
```

```
-----END CERTIFICATE-----
```

`/etc/pki/tls/certs/server.key`

Creates the private key file on the instance. Replace *`private key contents`* with the contents of the private key used to create the certificate request or self-signed certificate.

- The `container_commands` key restarts the nginx server after everything is configured so that the server uses the new `https.conf` file.

Example .ebextensions/https-instance.config

```
files:
  /etc/nginx/conf.d/https.conf:
    content: |
      # HTTPS server

      server {
          listen       443;
          server_name  localhost;

          ssl                  on;
          ssl_certificate      /etc/pki/tls/certs/server.crt;
          ssl_certificate_key  /etc/pki/tls/certs/server.key;

          ssl_session_timeout  5m;

          ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
          ssl_prefer_server_ciphers   on;

          location / {
              proxy_pass  http://my_app;
              proxy_set_header        Host            $host;
              proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
              proxy_set_header        X-Forwarded-Proto https;
          }

          location /assets {
            alias /var/app/current/public/assets;
            gzip_static on;
            gzip on;
            expires max;
            add_header Cache-Control public;
          }

          location /public {
            alias /var/app/current/public;
            gzip_static on;
            gzip on;
            expires max;
            add_header Cache-Control public;
          }
        }

  /etc/pki/tls/certs/server.crt:
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----

container_commands:
  01restart_nginx:
    command: "service nginx restart"
```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see Storing private keys securely in Amazon S3.

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an AWS CloudFormation function and adds a rule to it.

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load-balanced environment, you configure the load balancer to either pass secure traffic through untouched, or decrypt and re-encrypt for end-to-end encryption.

Configure HTTPS for Ruby with Passenger

For Ruby container types that use Passenger as the application server, you use both a configuration file and a JSON file to enable HTTPS.

To configure HTTPS for Ruby with Passenger

1. Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

   - The `files` key creates the following files on the instance:

     `/etc/pki/tls/certs/server.crt`

       Creates the certificate file on the instance. Replace *`certificate file contents`* with the contents of your certificate.

       Note

       YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

       If you have intermediate certificates, include them in `server.crt` after your site certificate.

       ```
             -----BEGIN CERTIFICATE-----
       ```

```
certificate file contents
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
first intermediate certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

`/etc/pki/tls/certs/server.key`

Creates the private key file on the instance. Replace *`private key contents`* with the contents of the private key used to create the certificate request or self-signed certificate.

Example .Ebextensions snippet for configuring HTTPS for Ruby with Passenger

```
files:
  /etc/pki/tls/certs/server.crt:
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----
```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see Storing private keys securely in Amazon S3.

2. Create a text file and add the following JSON to the file. Save it in your source bundle's root directory with the name `passenger-standalone.json`. This JSON file configures Passenger to use HTTPS.

Important

This JSON file must not contain a byte order mark (BOM). If it does, the Passenger JSON library will not read the file correctly and the Passenger service will not start.

Example passenger-standalone.json

```
{
  "ssl" : true,
  "ssl_port" : 443,
  "ssl_certificate" : "/etc/pki/tls/certs/server.crt",
  "ssl_certificate_key" : "/etc/pki/tls/certs/server.key"
}
```

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an AWS CloudFormation function and adds a rule to it.

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load-balanced environment, you configure the load balancer to either pass secure traffic through untouched, or decrypt and re-encrypt for end-to-end encryption.

Terminating HTTPS on EC2 instances running Tomcat

For Tomcat container types, you use a configuration file to enable the Apache HTTP Server to use HTTPS when acting as the reverse proxy for Tomcat.

Add the following snippet to your configuration file, replacing the certificate and private key material as instructed, and save it in your source bundle's `.ebextensions` directory. The configuration file performs the following tasks:

- The `files` key creates the following files on the instance:

  `/etc/pki/tls/certs/server.crt`

  Creates the certificate file on the instance. Replace *`certificate file contents`* with the contents of your certificate.

  > **Note**
  >
  > YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

  `/etc/pki/tls/certs/server.key`

  Creates the private key file on the instance. Replace *`private key contents`* with the contents of the private key used to create the certificate request or self-signed certificate.

  `/opt/elasticbeanstalk/hooks/appdeploy/post/99_start_httpd.sh`

  Creates a post-deployment hook script to restart the httpd service.

Example .ebextensions/https-instance.config

```
files:
  /etc/pki/tls/certs/server.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----

  /opt/elasticbeanstalk/hooks/appdeploy/post/99_start_httpd.sh:
    mode: "000755"
    owner: root
    group: root
    content: |
      #!/usr/bin/env bash
      sudo service httpd restart
```

You must also configure your environment's proxy server to listen on port 443. The following Apache 2.4 configuration adds a listener on port 443. To learn more, see Configuring the proxy server.

Example .ebextensions/httpd/conf.d/ssl.conf

```
Listen 443
<VirtualHost *:443>
  ServerName server-name
  SSLEngine on
  SSLCertificateFile "/etc/pki/tls/certs/server.crt"
  SSLCertificateKeyFile "/etc/pki/tls/certs/server.key"

  <Proxy *>
    Require all granted
  </Proxy>
  ProxyPass / http://localhost:8080/ retry=0
  ProxyPassReverse / http://localhost:8080/
  ProxyPreserveHost on

  ErrorLog /var/log/httpd/elasticbeanstalk-ssl-error_log

</VirtualHost>
```

Your certificate vendor may include intermediate certificates that you can install for better compatibility with mobile clients. Configure Apache with an intermediate certificate authority (CA) bundle by adding the following to your SSL configuration file (see Extending and overriding the default Apache configuration — Amazon Linux AMI (AL1) for the location):

- In the `ssl.conf` file contents, specify the chain file:

```
SSLCertificateKeyFile "/etc/pki/tls/certs/server.key"
SSLCertificateChainFile "/etc/pki/tls/certs/gd_bundle.crt"
SSLCipherSuite        EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
```

- Add a new entry to the `files` key with the contents of the intermediate certificates:

```
files:
  /etc/pki/tls/certs/gd_bundle.crt:
    mode: "000400"
    owner: root
    group: root
    content: |
      -----BEGIN CERTIFICATE-----
      First intermediate certificate
      -----END CERTIFICATE-----
      -----BEGIN CERTIFICATE-----
      Second intermediate certificate
      -----END CERTIFICATE-----
```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see Storing private keys securely in Amazon S3.

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an AWS CloudFormation function and adds a rule to it.

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load-balanced environment, you configure the load balancer to either pass secure traffic through untouched, or decrypt and re-encrypt for end-to-end encryption.

Terminating HTTPS on Amazon EC2 instances running .NET Core on Linux

For .NET Core on Linux container types, you enable HTTPS with an `.ebextensions` configuration file, and an nginx configuration file that configures the nginx server to use HTTPS.

Add the following snippet to your configuration file, replacing the certificate and private key placeholders as instructed, and save it in the `.ebextensions` directory. The configuration file performs the following tasks:

- The `files` key creates the following files on the instance:

  `/etc/pki/tls/certs/server.crt`

  Creates the certificate file on the instance. Replace *`certificate file contents`* with the contents of your certificate.

  > **Note**
  >
  > YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

  If you have intermediate certificates, include them in `server.crt` after your site certificate.

  ```
        -----BEGIN CERTIFICATE-----
    certificate file contents
    -----END CERTIFICATE-----
    -----BEGIN CERTIFICATE-----
    first intermediate certificate
    -----END CERTIFICATE-----
  ```

```
-----BEGIN CERTIFICATE-----
second intermediate certificate
-----END CERTIFICATE-----
```

`/etc/pki/tls/certs/server.key`

Creates the private key file on the instance. Replace *`private key contents`* with the contents of the private key used to create the certificate request or self-signed certificate.

- The `container_commands` key restarts the nginx server after everything is configured so that the server loads the nginx configuration file.

Example .ebextensions/https-instance.config

```
files:
  /etc/pki/tls/certs/server.crt:
    content: |
      -----BEGIN CERTIFICATE-----
      certificate file contents
      -----END CERTIFICATE-----

  /etc/pki/tls/certs/server.key:
    content: |
      -----BEGIN RSA PRIVATE KEY-----
      private key contents # See note below.
      -----END RSA PRIVATE KEY-----

container_commands:
  01restart_nginx:
    command: "systemctl restart nginx"
```

Note

Avoid committing a configuration file that contains your private key to source control. After you have tested the configuration and confirmed that it works, store your private key in Amazon S3 and modify the configuration to download it during deployment. For instructions, see Storing private keys securely in Amazon S3.

Place the following in a file with the `.conf` extension in the `.platform/nginx/conf.d/` directory of your source bundle (for example, `.platform/nginx/conf.d/https.conf`). Replace *app_port* with the port number that your application listens on. This example configures the nginx server to listen on port 443 using SSL. For more information about these configuration files on the .NET Core on Linux platform, see the section called “Proxy server”.

Example .platform/nginx/conf.d/https.conf

```
# HTTPS server

server {
    listen       443 ssl;
    server_name  localhost;

    ssl_certificate      /etc/pki/tls/certs/server.crt;
    ssl_certificate_key  /etc/pki/tls/certs/server.key;

    ssl_session_timeout  5m;

    ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers   on;

    location / {
        proxy_pass  http://localhost:app_port;
        proxy_set_header   Connection "";
        proxy_http_version 1.1;
        proxy_set_header        Host            $host;
        proxy_set_header        X-Real-IP       $remote_addr;
        proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header        X-Forwarded-Proto https;
    }
}
```

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an AWS CloudFormation function and adds a rule to it.

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
```

```
        IpProtocol: tcp
        ToPort: 443
        FromPort: 443
        CidrIp: 0.0.0.0/0
```

For a load-balanced environment, you configure the load balancer to either pass secure traffic through untouched, or decrypt and re-encrypt for end-to-end encryption.

Terminating HTTPS on Amazon EC2 instances running .NET

The following configuration file creates and runs a Windows PowerShell script that performs the following tasks:

- Checks for an existing HTTPS certificate binding to port 443.
- Gets the PFX certificate from an Amazon S3 bucket.

  Note

  Add an `AmazonS3ReadOnlyAccess` policy to the `aws-elasticbeanstalk-ec2-role` to access the SSL certificate in the Amazon S3 bucket.

- Gets the password from AWS Secrets Manager.

  Note

  Add a statement in `aws-elasticbeanstalk-ec2-role` that allows the `secretsmanager:GetSecretValue` action for the secret that contains the certificate password

- Installs the certificate.
- Binds the certificate to port 443.

  Note

  To remove the HTTP endpoint (port 80), include the `Remove-WebBinding` command under the **Remove the HTTP binding** section of the example.

Example .ebextensions/https-instance-dotnet.config

```
files:
  "C:\\certs\\install-cert.ps1":
    content: |
      import-module webadministration
      ## Settings - replace the following values with your own
      $bucket = "amzn-s3-demo-bucket"  ## S3 bucket name
      $certkey = "example.com.pfx"    ## S3 object key for your PFX certificate
      $secretname = "example_secret"  ## AWS Secrets Manager name for a secret that
 contains the certificate's password
      ##

      # Set variables
      $certfile = "C:\cert.pfx"
      $pwd = Get-SECSecretValue -SecretId $secretname | select -expand SecretString

      # Clean up existing binding
      if ( Get-WebBinding "Default Web Site" -Port 443 ) {
        Echo "Removing WebBinding"
        Remove-WebBinding -Name "Default Web Site" -BindingInformation *:443:
      }
      if ( Get-Item -path IIS:\SslBindings\0.0.0.0!443 ) {
        Echo "Deregistering WebBinding from IIS"
        Remove-Item -path IIS:\SslBindings\0.0.0.0!443
      }

      # Download certificate from S3
      Read-S3Object -BucketName $bucket -Key $certkey -File $certfile

      # Install certificate
      Echo "Installing cert..."
      $securepwd = ConvertTo-SecureString -String $pwd -Force -AsPlainText
      $cert = Import-PfxCertificate -FilePath $certfile cert:\localMachine\my -Password
 $securepwd

      # Create site binding
      Echo "Creating and registering WebBinding"
      New-WebBinding -Name "Default Web Site" -IP "*" -Port 443 -Protocol https
      New-Item -path IIS:\SslBindings\0.0.0.0!443 -value $cert -Force

      ## Remove the HTTP binding
      ## (optional) Uncomment the following line to unbind port 80
      # Remove-WebBinding -Name "Default Web Site" -BindingInformation *:80:
```

```
      ##

      # Update firewall
      netsh advfirewall firewall add rule name="Open port 443" protocol=TCP
 localport=443 action=allow dir=OUT

commands:
  00_install_ssl:
    command: powershell -NoProfile -ExecutionPolicy Bypass -file C:\\certs\\install-
cert.ps1
```

In a single instance environment, you must also modify the instance's security group to allow traffic on port 443. The following configuration file retrieves the security group's ID using an AWS CloudFormation function and adds a rule to it.

Example .ebextensions/https-instance-single.config

```
Resources:
  sslSecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      CidrIp: 0.0.0.0/0
```

For a load-balanced environment, you configure the load balancer to either pass secure traffic through untouched, or decrypt and re-encrypt for end-to-end encryption.

Configuring end-to-end encryption in a load-balanced Elastic Beanstalk environment

Terminating secure connections at the load balancer and using HTTP on the backend might be sufficient for your application. Network traffic between AWS resources can't be listened to by instances that are not part of the connection, even if they are running under the same account.

However, if you are developing an application that needs to comply with strict external regulations, you might be required to secure all network connections. You can use the Elastic Beanstalk console or configuration files to make your Elastic Beanstalk environment's load balancer connect to backend instances securely to meet these requirements. The following procedure focuses on configuration files.

First, add a secure listener to your load balancer, if you haven't already.

You must also configure the instances in your environment to listen on the secure port and terminate HTTPS connections. The configuration varies per platform. See Configuring HTTPS Termination at the instance for instructions. You can use a self-signed certificate for the EC2 instances without issue.

Next, configure the listener to forward traffic using HTTPS on the secure port used by your application. Use one of the following configuration files, based on the type of load balancer that your environment uses.

`.ebextensions/https-reencrypt-clb.config`

Use this configuration file with a Classic Load Balancer. In addition to configuring the load balancer, the configuration file also changes the default health check to use port 443 and HTTPS, to ensure that the load balancer can connect securely.

```
option_settings:
  aws:elb:listener:443:
    InstancePort: 443
    InstanceProtocol: HTTPS
  aws:elasticbeanstalk:application:
    Application Healthcheck URL: HTTPS:443/
```

`.ebextensions/https-reencrypt-alb.config`

Use this configuration file with an Application Load Balancer.

```
option_settings:
  aws:elbv2:listener:443:
    DefaultProcess: https
    ListenerEnabled: 'true'
    Protocol: HTTPS
  aws:elasticbeanstalk:environment:process:https:
    Port: '443'
    Protocol: HTTPS
```

`.ebextensions/https-reencrypt-nlb.config`

Use this configuration file with a Network Load Balancer.

```
option_settings:
  aws:elbv2:listener:443:
    DefaultProcess: https
    ListenerEnabled: 'true'
  aws:elasticbeanstalk:environment:process:https:
    Port: '443'
```

The `DefaultProcess` option is named this way because of Application Load Balancers, which can have nondefault listeners on the same port for traffic to specific paths (see Application Load Balancer for details). For a Network Load Balancer the option specifies the only target process for this listener.

In this example, we named the process `https` because it listens to secure (HTTPS) traffic. The listener sends traffic to the process on the designated port using the TCP protocol, because a Network Load Balancer works only with TCP. This is okay, because network traffic for HTTP and HTTPS is implemented on top of TCP.

Note

The EB CLI and Elastic Beanstalk console apply recommended values for the preceding options. You must remove these settings if you want to use configuration files to configure the same. See Recommended values for details.

In the next task, you need to modify the load balancer's security group to allow traffic. Depending on the Amazon Virtual Private Cloud (Amazon VPC) in which you launch your environment—the default VPC or a custom VPC—the load balancer's security group will vary. In a default VPC, Elastic Load Balancing provides a default security group that all load balancers can use. In an Amazon VPC that you create, Elastic Beanstalk creates a security group for the load balancer to use.

To support both scenarios, you can create a security group and tell Elastic Beanstalk to use it. The following configuration file creates a security group and attaches it to the load balancer.

`.ebextensions/https-lbsecuritygroup.config`

```
option_settings:
  # Use the custom security group for the load balancer
  aws:elb:loadbalancer:
    SecurityGroups: '`{ "Ref" : "loadbalancersg" }`'
    ManagedSecurityGroup: '`{ "Ref" : "loadbalancersg" }`'

Resources:
  loadbalancersg:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: load balancer security group
      VpcId: vpc-########
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 443
          ToPort: 443
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
      SecurityGroupEgress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
```

Replace the highlighted text with your default or custom VPC ID. The previous example includes ingress and egress over port 80 to allow HTTP connections. You can remove those properties if you want to allow only secure connections.

Finally, add ingress and egress rules that allow communication over port 443 between the load balancer's security group and the instances' security group.

`.ebextensions/https-backendsecurity.config`

```
Resources:
  # Add 443-inbound to instance security group (AWSEBSecurityGroup)
  httpsFromLoadBalancerSG:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      SourceSecurityGroupId: {"Fn::GetAtt" : ["loadbalancersg", "GroupId"]}
  # Add 443-outbound to load balancer security group (loadbalancersg)
  httpsToBackendInstances:
    Type: AWS::EC2::SecurityGroupEgress
    Properties:
      GroupId: {"Fn::GetAtt" : ["loadbalancersg", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      DestinationSecurityGroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
```

Doing this separately from security group creation enables you to restrict the source and destination security groups without creating a circular dependency.

After you have completed all the previous tasks, the load balancer connects to your backend instances securely using HTTPS. The load balancer doesn't care if your instance's certificate is self-signed or issued by a trusted certificate authority, and will accept any certificate presented to it.

You can change this behavior by adding policies to the load balancer that tell it to trust only a specific certificate. The following configuration file creates two policies. One policy specifies a public certificate, and the other tells the load balancer to only trust that certificate for connections to instance port 443.

.ebextensions/https-backendauth.config

```
option_settings:
  # Backend Encryption Policy
  aws:elb:policies:backendencryption:
    PublicKeyPolicyNames: backendkey
    InstancePorts:  443
  # Public Key Policy
  aws:elb:policies:backendkey:
    PublicKey: |
      -----BEGIN CERTIFICATE-----
      ################################################################
      ################################################################
      ################################################################
      ################################################################
      ################################################
      -----END CERTIFICATE-----
```

Replace the highlighted text with the contents of your EC2 instance's public certificate.

Configuring your environment's load balancer for TCP Passthrough

If you do not want the load balancer in your AWS Elastic Beanstalk environment to decrypt the HTTPS traffic, you can configure the secure listener to relay requests to backend instances as-is.

Important

Configuring the load balancer to relay HTTPS traffic without decrypting it presents a disadvantage. The load balancer cannot see the encrypted requests and thus cannot optimize routing or report response metrics.

First configure your environment's EC2 instances to terminate HTTPS. Test the configuration on a single instance environment to make sure everything works before adding a load balancer to the mix.

Add a configuration file to your project to configure a listener on port 443 that passes TCP packets as-is to port 443 on backend instances:

`.ebextensions/https-lb-passthrough.config`

```
option_settings:
  aws:elb:listener:443:
    ListenerProtocol: TCP
    InstancePort: 443
    InstanceProtocol: TCP
```

In a default Amazon Virtual Private Cloud (Amazon VPC), you also need to add a rule to the instances' security group to allow inbound traffic on 443 from the load balancer:

`.ebextensions/https-instance-securitygroup.config`

```
Resources:
  443inboundfromloadbalancer:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 443
      FromPort: 443
      SourceSecurityGroupName: { "Fn::GetAtt": ["AWSEBLoadBalancer",
"SourceSecurityGroup.GroupName"] }
```

In a custom VPC, Elastic Beanstalk updates the security group configuration for you.

Configuring HTTP to HTTPS redirection

This topic describes how to handle HTTP traffic to your application if end users still initiate it. You do this by configuring *HTTP to HTTPS redirection*, sometimes referred to as *forcing HTTPS*.

To configure redirection, you first configure your environment to handle HTTPS traffic. Then you redirect HTTP traffic to HTTPS. These two steps are discussed in the following subsections.

Configure your environment to handle HTTPS traffic

Depending on your environment's load balancing configuration, do one of the following:

- **Load-balanced environment –** Configure your load balancer to terminate HTTPS.
- **Single-instance environment –** Configure your application to terminate HTTPS connections at the instance. This configuration depends on your environment's platform.

Redirect HTTP traffic to HTTPS

To redirect HTTP traffic to HTTPS for your application you can either configure the web servers on your environment's instances or you can configure the environment's Application Load Balancer.

Configure the instance web servers

This method works on any web server environment. Configure web servers on your Amazon EC2 instances to respond to HTTP traffic with an HTTP redirection response status.

This configuration depends on your environment's platform. Find the folder for your platform in the `https-redirect` collection on GitHub, and use the example configuration file in that folder.

If your environment uses Elastic Load Balancing health checks, the load balancer expects a healthy instance to respond to the HTTP health check messages with HTTP 200 (OK) responses. Therefore, your web server shouldn't redirect these messages to HTTPS. The example configuration files in `https-redirect` handle this requirement correctly.

Configure the load balancer

This method works if you have a load-balanced environment that uses an Application Load Balancer. An Application Load Balancer can send redirection responses as HTTP traffic comes in. In this case, you don't need to configure redirection on your environment's instances.

We have two example configuration files on GitHub that show how to configure an Application Load Balancer for redirection.

- The `alb-http-to-https-redirection-full.config` configuration file creates an HTTPS listener on port 443, and modifies the default port 80 listener to redirect incoming HTTP traffic to HTTPS.
- The `alb-http-to-https-redirection.config` configuration file expects the 443 listener to be defined. To define it, you can use standard Elastic Beanstalk configuration namespaces, or the Elastic Beanstalk console. Then it takes care of modifying the port 80 listener for redirection.

Elastic Beanstalk platforms

AWS Elastic Beanstalk provides a variety of platforms on which you can build your applications. You design your web application to one of these platforms, and Elastic Beanstalk deploys your code to the platform version you selected to create an active application environment.

Elastic Beanstalk provides platforms for different programming languages, application servers, and Docker containers. Some platforms have multiple concurrently-supported versions.

Topics

Elastic Beanstalk platforms glossary

Following are key terms related to AWS Elastic Beanstalk platforms and their lifecycle.

Runtime

The programming language-specific runtime software (framework, libraries, interpreter, vm, etc.) required to run your application code.

Elastic Beanstalk Components

Software components that Elastic Beanstalk adds to a platform to enable Elastic Beanstalk functionality. For example, the enhanced health agent is necessary for gathering and reporting health information.

Platform

A combination of an operating system (OS), runtime, web server, application server, and Elastic Beanstalk components. Platforms provide components that are available to run your application.

Platform Version

A combination of specific versions of an operating system (OS), runtime, web server, application server, and Elastic Beanstalk components. You create an Elastic Beanstalk environment based on a platform version and deploy your application to it.

A platform version has a semantic version number of the form *X.Y.Z*, where *X* is the major version, *Y* is the minor version, and *Z* is the patch version.

A platform version can be in one of the following states:

- *Recommended* – The latest platform version in a supported platform branch. This version contains the most up-to-date components and is recommended for use in production environments. When Elastic Beanstalk releases a new platform version, the new version supersedes the previous version and becomes the recommended platform version for the corresponding platform branch.
- *Not Recommended* – Any platform version that is not the latest version in its platform branch. While these versions may remain functional, we strongly recommend updating to the latest platform version. You can use managed platform updates to help stay up-to-date automatically.

You can verify if a platform version is recommended using the AWS CLI command **describe-platform-version** and checking the `PlatformLifecycleState` field.

Platform Branch

A line of platform versions sharing specific (typically major) versions of some of their components, such as the operating system (OS), runtime, or Elastic Beanstalk components. For example: *Python 3.13 running on 64bit Amazon Linux 2023*; *IIS 10.0 running on 64bit Windows Server 2025*. Platform branches receive updates in the form of new platform versions. Each successive platform version in a branch is an update to the previous one.

The recommended version in each supported platform branch is available to you unconditionally for environment creation. Previous platform versions remain accessible to accounts with active or terminated environments using them at the time they were superseded by a new version. Previous platform versions lack the most up-to-date components and aren't recommended for use.

A platform branch can be in one of the following states:

- *Supported* – A current platform branch. It consists entirely of *supported components*. Supported components have not reached End of Life (EOL), as designated by their suppliers. It receives ongoing platform updates, and is recommended for use in production environments. For a list of supported platform branches, see Elastic Beanstalk supported platforms in the *AWS Elastic Beanstalk Platforms* guide.

- *Beta* – A preview, pre-release platform branch. It's experimental in nature. It may receive ongoing platform updates for a while, but has no long-term support. A beta platform branch isn't recommended for use in production environments. Use it only for evaluation. For a list of beta platform branches, see Elastic Beanstalk Platform Versions in Public Beta in the *AWS Elastic Beanstalk Platforms* guide.
- *Deprecated* – A platform branch where one or more components (such as the runtime or operating system) are approaching End of Life (EOL) or have reached EOL, as designated by their suppliers. While a deprecated platform branch continues to receive new platform versions until its retirement date, components that have reached EOL don't receive updates. For example, if a runtime version reaches EOL, the platform branch will be marked as deprecated but will continue to receive operating system updates until the platform branch retirement date. The platform branch will not continue to receive updates to the EOL runtime version. A deprecated platform branch isn't recommended for use.
- *Retired* – A platform branch that no longer receives any updates. Retired platform branches aren't available to create new Elastic Beanstalk environments using the Elastic Beanstalk console. If your environment uses a retired platform branch, you must update to a supported platform branch to continue receiving updates. A retired platform branch isn't recommended for use. For more details about retired platform branches, see the section called “Platform support policy”. For a list of platform branches scheduled for retirement, see Retiring platform branch schedule. To see past retired platform branches, see Retired platform branch history.

If your environment uses a deprecated or retired platform branch, we recommend that you update it to a platform version in a supported platform branch. For details, see the section called “Platform updates”.

You can verify the state of a platform branch using the AWS CLI command **describe-platform-version** and checking the `PlatformBranchLifecycleState` field.

Platform Update

A release of a new platform version that contains updates to some components of the platform—OS, runtime, web server, application server, and Elastic Beanstalk components. When Elastic Beanstalk releases a new platform version, the new version supersedes the previous version and becomes the recommended platform version for the corresponding platform branch. Platform updates follow semantic version taxonomy, and can have three levels:

- *Major update* – An update that has changes that are incompatible with existing platform versions. You may need to modify your application to run correctly on a new major version. A major update has a new major platform version number.
- *Minor update* – An update that has changes that are backward compatible with existing platform versions in most cases. Depending on your application, you may need to modify your application to run correctly on a new minor version. A minor update has a new minor platform version number.
- *Patch update* – An update that consists of maintenance releases (bug fixes, security updates, and performance improvements) that are backward compatible with an existing platform version. A patch update has a new patch platform version number.

Managed Updates

An Elastic Beanstalk feature that automatically applies patch and minor updates to the operating system (OS), runtime, web server, application server, and Elastic Beanstalk components for an Elastic Beanstalk supported platform version. A managed update applies a newer platform version in the same platform branch to your environment. You can configure managed updates to apply only patch updates, or minor and patch updates. You can also disable managed updates completely.

For more information, see Managed platform updates.

Shared responsibility model for Elastic Beanstalk platform maintenance

AWS and our customers share responsibility for achieving a high level of software component security and compliance. This shared model reduces your operational burden.

For details, see the AWS Shared Responsibility Model.

AWS Elastic Beanstalk helps you perform your side of the shared responsibility model by providing a *managed updates* feature. This feature automatically applies patch and minor updates for an Elastic Beanstalk supported platform version. If a managed update fails, Elastic Beanstalk notifies you of the failure to ensure that you are aware of it and can take immediate action.

For more information, see Managed platform updates.

In addition, Elastic Beanstalk does the following:

- Publishes its platform support policy and retirement schedule for the coming 12 months.
- Releases patch, minor, and major updates of operating system (OS), runtime, application server, and web server components typically within 30 days of their availability. Elastic Beanstalk is responsible for creating updates to Elastic Beanstalk components that are present on its supported platform versions. All other updates come directly from their suppliers (owners or community).

We announce all updates to our supported platforms in our release notes in the *AWS Elastic Beanstalk Release Notes* guide. We also provide a list of all supported platforms and their components, along with a platform history, in the *AWS Elastic Beanstalk Platforms* guide. For more information see Supported platforms and component history.

You are responsible to do the following:

- Update all the components that you control (identified as **Customer** in the AWS Shared Responsibility Model). This includes ensuring the security of your application, your data, and any components that your application requires and that you downloaded.
- Ensure that your Elastic Beanstalk environments are running on a supported platform version, and migrate any environment running on a retired platform version to a supported version.
- If you're using a custom Amazon machine image (AMI) for your Elastic Beanstalk environment, patch, maintain, and test your custom AMI so that it remains current and compatible with a supported Elastic Beanstalk platform version. For more information about managing environments with a custom AMI, see Using a custom Amazon machine image (AMI) in your Elastic Beanstalk environment.
- Resolve all issues that come up in failed managed update attempts and retry the update.
- Patch the OS, runtime, application server, and web server yourself if you opted out of Elastic Beanstalk managed updates. You can do this by applying platform updates manually or directly patching the components on all relevant environment resources.
- Manage the security and compliance of any AWS services that you use outside of Elastic Beanstalk according to the AWS Shared Responsibility Model.

Elastic Beanstalk platform support policy

Elastic Beanstalk supports platform branches that still receive ongoing minor and patch updates from their suppliers (owners or community). For a complete definition of related terms, see Elastic Beanstalk platforms glossary.

Retired platform branches

When a component of a supported platform branch is marked End of Life (EOL) by its supplier, Elastic Beanstalk marks the platform branch as retired. Components of a platform branch include the following: operating system (OS), runtime language version, application server, or web server.

Once a platform branch is marked as retired the following policies apply:

- Elastic Beanstalk stops providing maintenance updates, including security updates.
- Elastic Beanstalk no longer provides technical support for retired platform branches.
- Elastic Beanstalk no longer makes the platform branch available to new Elastic Beanstalk customers for deployments to new environments. There is a 90 day grace period from the published retirement date for existing customers with active environments that are running on retired platform branches.

Note

A retired platform branch will not be available in the Elastic Beanstalk console. However, it will be available through the AWS CLI, EB CLI and EB API for customers that have existing environments based on the retired platform branch. Existing customers can also use the Clone environment and Rebuild environment consoles.

For a list of platform branches that are scheduled for retirement see the Retiring platform branch schedule in the *Elastic Beanstalk platform schedule* topic that follows.

For more information about what to expect when your environment's platform branch retires, see Platform retirement FAQ.

Beyond the 90 day grace period

Our policy for retired platform branches does not remove access to environments nor delete resources. However, existing customers running an Elastic Beanstalk environment on a retired platform branch should be aware of the risks of doing so. Such environments can end up in an unpredictable situation, because Elastic Beanstalk isn't able to provide security updates, technical support, or hotfixes for retired platform branches due to the supplier marking their component EOL.

For example, a detrimental and critical security vulnerability may surface in an environment running on a retired platform branch. Or an EB API action may stop working for the environment if it becomes incompatible with the Elastic Beanstalk service over time. The opportunity for these types of risks increases the longer an environment on a retired platform branch remains active. To continue to benefit from important security, performance, and functionality enhancements offered by component suppliers in more recent releases, we strongly encourage you to update all your Elastic Beanstalk environments to a supported platform version.

If your application should encounter issues while running on a retired platform branch and you're not able to migrate it to a supported platform, you'll need to consider other alternatives. Workarounds include encapsulating the application into a Docker image to run it as a Docker container. This would allow a customer to use any of our Docker solutions, such as our Elastic Beanstalk AL2023/AL2 Docker platforms, or other Docker based services such as Amazon ECS or Amazon EKS. Non-Docker alternatives include our AWS CodeDeploy service, which allows complete customization of the runtimes you desire.

Elastic Beanstalk platform release schedule

In addition to the monthly cadence release of new platform branch versions, our release maintenance also includes the following processes:

- *Release of new platform branches* – These typically introduce a new major version of a run-time language, operating system or application server.
- *Retirement of platform branches* – We must retire a platform branch when one of its components reaches End of Life (EOL). For more information about our policy for retired branches, see Elastic Beanstalk platform support policy

Topics

- Planning resources
- Upcoming platform branch releases
- Retiring platform branch schedule
- Retired platform branch history
- Retired server and operating system history

Planning resources

The following resources can help you plan maintenance and support for your application running on an Elastic Beanstalk platform.

- AWS Elastic Beanstalk Platforms guide — This guide provides a detailed component list for each of our platform branches. It also provides a platform history by release date with the same details. This guide can inform you when specific components of your platform branch changed. If your application starts behaving differently, you can also cross-reference the date of the occurrence in the platforms guide to see if there were any platform changes that might have affected your application.
- AWS Elastic Beanstalk Release Notes — Our Release Notes announce all of our platform releases, both minor and major. This includes our monthly platform updates, security releases, hotfixes, and retirement announcements. You can subscribe to our RSS feeds from the Release Notes documentation.

Upcoming platform branch releases

The following table lists upcoming Elastic Beanstalk platform branches and their target release date. These dates are tentative and subject to change.

Runtime version / platform branch	Operating System	Target release date
Corretto 25	Amazon Linux 2023	October 2025
Corretto 25 with Tomcat 11	Amazon Linux 2023	October 2025
Python 3.14	Amazon Linux 2023	November 2025

Runtime version / platform branch	Operating System	Target release date
Node.js 24	Amazon Linux 2023	November 2025
.NET 10	Amazon Linux 2023	December 2025
PHP 8.5	Amazon Linux 2023	January 2026
Ruby 3.5	Amazon Linux 2023	February 2026

Retiring platform branch schedule

This following table lists Elastic Beanstalk platform branches that are scheduled for retirement, because some of their components are reaching their End of Life (EOL). All AL2-based platform branches have retirement dates no later than June 30, 2026, when Amazon Linux 2 reaches EOL. For more information about Amazon Linux 2, see Amazon Linux 2 FAQs.

For a more detailed list of retiring platform branches that includes their specific components, see retiring platform versions in the *AWS Elastic Beanstalk Platforms* guide.

Runtime version / platform branch	Target retirement date	
PHP 8.1 AL2023	March 31, 2026	
PHP 8.1 AL2	March 31, 2026	
Docker AL2	June 30, 2026	
ECS AL2	June 30, 2026	
Go 1 AL2	June 30, 2026	
Corretto 8 AL2	June 30, 2026	
Corretto 11 AL2	June 30, 2026	
Corretto 17 AL2	June 30, 2026	

Runtime version / platform branch	Target retirement date	
Corretto 8 with Tomcat 9 AL2	June 30, 2026	
Corretto 11 with Tomcat 9 AL2	June 30, 2026	
.NET Core AL2	June 30, 2026	
Python 3.9 AL2023	July 31, 2026	
Ruby 3.2 AL2023	July 31, 2026	
Node.js 20 AL2023	July 31, 2026	
.NET 9 AL2023	July 31, 2026	
IIS 10.0 on Windows Server 2016 (& Core)	September 30, 2026	

Retired platform branch history

The following tables list Elastic Beanstalk platform branches that are already in retired status. You can see a detailed history of these platform branches and their components in the Platform history of the *AWS Elastic Beanstalk Platforms* guide.

Amazon Linux 2023 (AL2023)

Runtime version / platform branch	Retirement date		
.NET 6 AL2023	April 8, 2025		
Node.js 18 AL2023	August 11, 2025		

Amazon Linux 2 (AL2)

Runtime version / platform branch	Retirement date		
Corretto 11 with Tomcat 8.5 AL2	October 10, 2024		
Corretto 8 with Tomcat 8.5 AL2	October 10, 2024		
Corretto 11 with Tomcat 7 AL2	June 29, 2022		
Corretto 8 with Tomcat 7 AL2	June 29, 2022		
Node.js 18 AL2	August 11, 2025		
Node.js 16 AL2	October 10, 2024		
Node.js 14 AL2	October 10, 2024		
Node.js 12 AL2	December 23, 2022		
Node.js 10 AL2	June 29, 2022		
PHP 8.0 AL2	October 10, 2024		
PHP 7.4 AL2	June 9, 2023		
PHP 7.3 AL2	June 29, 2022		
PHP 7.2 AL2	June 29, 2022		
Python 3.8 AL2	April 8, 2025		
Python 3.7 AL2	October 10, 2024		

Runtime version / platform branch	Retirement date		
Ruby 3.0 AL2	October 10, 2024		
Ruby 2.7 AL2	October 10, 2024		
Ruby 2.6 AL2	December 23, 2022		
Ruby 2.5 AL2	June 29, 2022		

Amazon Linux AMI (AL1)

Runtime version / platform branch	Retirement date		
Single Container Docker	July 18, 2022		
Multicontainer Docker	July 18, 2022		
Preconfig ured Docker - GlassFish 5.0 with Java 8	July 18, 2022		
Go 1	July 18, 2022		
Java 8	July 18, 2022		
Java 7	July 18, 2022		
Java 8 with Tomcat 8.5	July 18, 2022		

Runtime version / platform branch	Retirement date		
Java 7 with Tomcat 7	July 18, 2022		
Node.js	July 18, 2022		
PHP 7.2 - 7.3	July 18, 2022		
Python 3.6	July 18, 2022		
Ruby 2,4, 2.5, 2.6 with Passenger	July 18, 2022		
Ruby 2.4, 2.5, 2.6 with Puma	July 18, 2022		
Go 1.3–1.10	October 31, 2020		
Java 6	October 31, 2020		
Node.js 4.x–8.x	October 31, 2020		
PHP 5.4–5.6	October 31, 2020		
PHP 7.0–7.1	October 31, 2020		
Python 2.6, 2.7, 3.4	October 31, 2020		
Ruby 1.9.3	October 31, 2020		
Ruby 2.0–2.3	October 31, 2020		

Note

On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information, see Platform retirement FAQ.

Windows Server

Runtime version / platform branch	Retirement date		
IIS 8.5 running on 64bit Windows Server (& Core) 2012 R2	December 4, 2023		
IIS 8.5 running on 64bit Windows Server (& Core) 2012 R2 version 0.1.0	June 29, 2022		
IIS 8.5 running on 64bit Windows Server (& Core) 2012 R2 version 1.2.0	June 29, 2022		
IIS 10.0 running on 64bit Windows Server 2016 (& Core) version 1.2.0	June 29, 2022		
IIS 8 running on 64bit Windows	June 22, 2022		

Runtime version / platform branch	Retirement date
Server 2012 R1 Platform Branch	
IIS 8 running on 64bit Windows Server 2012 R1 version 0.1.0	June 22, 2022
IIS 8 running on 64bit Windows Server 2012 R1 version 1.2.0	June 22, 2022

Note

For more information about the retirement of the *Windows 2012 R2* platform branches, see Windows Server 2012 R2 platform branches retired in the *AWS Elastic Beanstalk Release Notes*.

Retired server and operating system history

The following tables provide a history of the operating systems, application servers, and web servers that are no longer supported by Elastic Beanstalk platforms. All of the platform branches that utilized these components are now retired. The dates reflect the retirement date of the last Elastic Beanstalk platform branch that included the component.

Operating Systems

OS version	Platform retirement date		
Windows Server 2012 R2 running IIS 8.5	December 4, 2023		
Windows Server Core 2012 R2 running IIS 8.5	December 4, 2023		
Amazon Linux AMI (AL1)	July 18, 2022		
Windows Server 2012 R1	June 22, 2022		
Windows Server 2008 R2	October 28, 2019		

Application servers

Application server version	Platform retirement date		
Tomcat 7	June 29, 2022 for Amazon Linux 2 (AL2) platforms July 18, 2022 for Amazon Linux AMI (AL1) platforms		
Tomcat 8	October 31, 2020		
Tomcat 6	October 31, 2020		

Web servers

Web server version	Platform retirement date		
IIS 8 running on 64bit Windows Server	June 22, 2022		
Apache HTTP Server 2.2	October 31, 2020		
Nginx 1.12.2	October 31, 2020		

Elastic Beanstalk supported platforms

AWS Elastic Beanstalk provides a variety of platforms on which you can build your applications. You design your web application to one of these platforms, and Elastic Beanstalk deploys your code to the platform version you selected to create an active application environment.

Elastic Beanstalk provisions the resources needed to run your application, including one or more Amazon EC2 instances. The software stack running on the Amazon EC2 instances depends on the specific platform version you've selected for your environment.

The solution stack name for a platform branch

You can use the *solution stack name* for a given platform branch version to launch an environment with the EB CLI, Elastic Beanstalk API, or the AWS CLI. The *AWS Elastic Beanstalk Platforms* guide lists the *solution stack name* under the platform branch version in both the Elastic Beanstalk Supported Platforms and Platform history sections.

To retrieve all of the solution stack names that you can use to create an environment, use the ListAvailableSolutionStacks API or the `aws elasticbeanstalk list-available-solution-stacks` in the AWS CLI.

You can customize and configure the software that your application depends on in your platform. Learn more at Customizing software on Linux servers and Customizing software on Windows servers. Detailed release notes are available for recent releases at AWS Elastic Beanstalk Release Notes.

Supported platforms and component history

The *AWS Elastic Beanstalk Platforms* guide lists all of the current platform branch versions in the Elastic Beanstalk Supported Platforms section. The *Platforms* guide also lists a *platform history* for each platform, which includes a list of previous branch platform versions. To view the *platform history* for each platform, select one of the following links.

- Docker
- Go
- Java SE
- Tomcat (running Java SE)
- .NET Core on Linux
- .NET on Windows Server
- Node.js
- PHP
- Python
- Ruby

Elastic Beanstalk Linux platforms

The Elastic Beanstalk Linux platforms provide an extensive amount of functionality out of the box. You can extend the platforms in several ways to support your application. For details, see the section called “Extending Linux platforms”.

Most of the platforms that Elastic Beanstalk supports are based on the Linux operating system. Specifically, these platforms are based on Amazon Linux, a Linux distribution provided by AWS. Elastic Beanstalk Linux platforms use Amazon Elastic Compute Cloud (Amazon EC2) instances, and these instances run Amazon Linux.

Topics

Supported Amazon Linux versions

AWS Elastic Beanstalk supports platforms based on Amazon Linux 2 and Amazon Linux 2023.

For more information about Amazon Linux 2 and Amazon Linux 2023, see the following:

- **Amazon Linux 2** – Amazon Linux in the *Amazon EC2 User Guide*.
- **Amazon Linux 2023** – What is Amazon Linux 2023? in the *Amazon Linux 2023 User Guide*

For details about supported platform versions, see Elastic Beanstalk supported platforms.

> **Note**
>
> You can migrate your application from an Elastic Beanstalk AL1 or AL2 platform branch to the equivalent AL2023 platform branch. For more information, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

Amazon Linux 2023

AWS announced the general availability of Amazon Linux 2023 in March of 2023. The *Amazon Linux 2023 User Guide* summarizes key differences between Amazon Linux 2 and Amazon Linux 2023. For more information, see Comparing Amazon Linux 2 and Amazon Linux 2023 in the user guide.

There is a high degree of compatibility between Elastic Beanstalk Amazon Linux 2 and Amazon Linux 2023 platforms. Although there are some differences to note:

- **Instance Metadata Service Version 1 (IMDSv1)** – The DisableIMDSv1 option setting defaults to `true` on AL2023 platforms. The default is `false` on AL2 platforms.
- **pkg-repo instance tool** – The pkg-repo tool is not available for environments running on AL2023 platforms. However,you can manually apply package and operating system updates to an AL2023 instance. For more information, see Managing packages and operating system updates in the *Amazon Linux 2023 User Guide*.
- **Apache HTTPd configuration** – The Apache `httpd.conf` file for AL2023 platforms has some configuration settings that are different from those for AL2:
  - Deny access to the server's entire file system by default. These settings are described in *Protect Server Files by Default* on the Apache website Security Tips page.

- Stop users from overriding security features you've configured. The configuration denies access to set up of `.htaccess` in all directories, except for those specifically enabled. This setting is described in *Protecting System Settings* on the Apache website Security Tips page. The Apache HTTP Server Tutorial: .htaccess files page states this setting may help improve performance.
- Deny access to files with name pattern `.ht*`. This setting prevents web clients from viewing `.htaccess` and `.htpasswd` files.

You can change any of the above configuration settings for your environment. For more information, see Configuring Apache HTTPD.

List of Elastic Beanstalk Linux platforms

The following list provides the Linux platforms that Elastic Beanstalk supports for different programming languages as well as for Docker containers. Elastic Beanstalk offers platforms based on Amazon Linux 2 and Amazon Linux 2023 for all of them. To learn more about a platform, select the corresponding link.

- Docker (and ECS Docker)
- Go
- Tomcat (running Java SE)
- Java SE
- .NET Core on Linux
- Node.js
- PHP
- Python
- Ruby

Instance deployment workflow

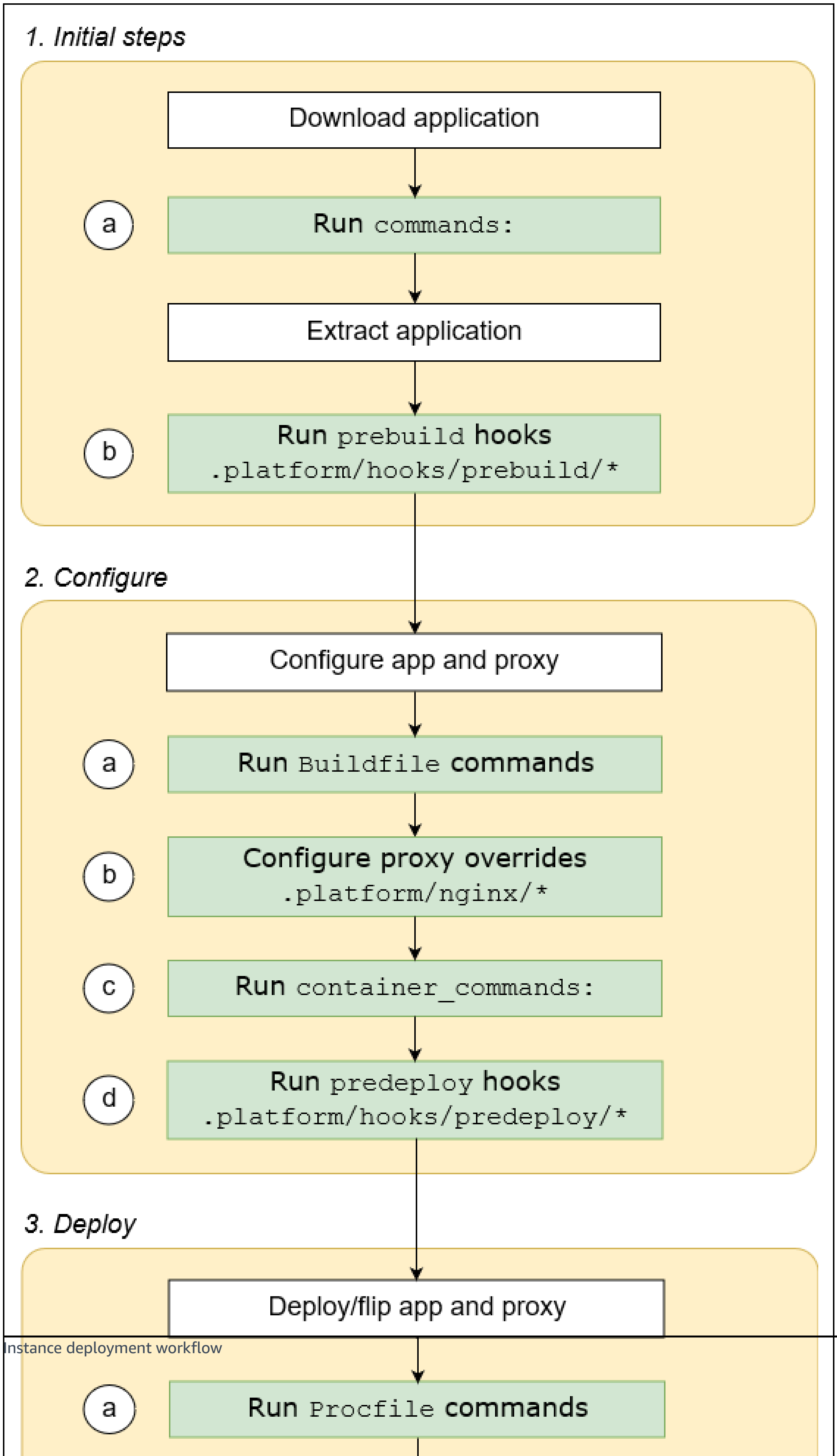
Note

The information in this section doesn't apply to the *ECS running on Amazon Linux 2 and Amazon Linux 2023* platform branches. For more information, see the next section Instance deployment workflow for ECS running on Amazon Linux 2 and later.

With many ways to extend your environment's platform, it's useful to know what happens whenever Elastic Beanstalk provisions an instance or runs a deployment to an instance. The following diagram shows this entire deployment workflow. It depicts the different phases in a deployment and the steps that Elastic Beanstalk takes in each phase.

Notes

- The diagram doesn't represent the complete set of steps that Elastic Beanstalk takes on environment instances during deployment. We provide this diagram for illustration, to provide you with the order and context for the execution of your customizations.
- For simplicity, the diagram mentions only the `.platform/hooks/*` hook subdirectories (for application deployments), and not the `.platform/confighooks/*` hook subdirectories (for configuration deployments). Hooks in the latter subdirectories run during exactly the same steps as hooks in corresponding subdirectories shown in the diagram.


1. Initial steps
Download application
a
Run commands:
Extract application
b
Run prebuild hooks
.platform/hooks/prebuild/*
2. Configure
Configure app and proxy
a
Run Buildfile commands
b
Configure proxy overrides
.platform/nginx/*
c
Run container_commands:
d
Run predeploy hooks
.platform/hooks/predeploy/*
3. Deploy
Deploy/flip app and proxy
a
Run Procfile commands

The following list details the deployment phases and steps.

1. **Initial steps**

   Elastic Beanstalk downloads and extracts your application. After each one of these steps, Elastic Beanstalk runs one of the extensibility steps.

   a. Runs commands found in the commands: section of any configuration file.

   b. Runs any executable files found in the `.platform/hooks/prebuild` directory of your source bundle (`.platform/confighooks/prebuild` for a configuration deployment).

2. **Configure**

   Elastic Beanstalk configures your application and the proxy server.

   a. Runs the commands found in the `Buildfile` in your source bundle.

   b. Copies your custom proxy configuration files, if you have any in the `.platform/nginx` directory of your source bundle, to their runtime location.

   c. Runs commands found in the container_commands: section of any configuration file.

   d. Runs any executable files found in the `.platform/hooks/predeploy` directory of your source bundle (`.platform/confighooks/predeploy` for a configuration deployment).

3. **Deploy**

   Elastic Beanstalk deploys and runs your application and the proxy server.

   a. Runs the command found in the `Procfile` file in your source bundle.

   b. Runs or reruns the proxy server with your custom proxy configuration files, if you have any.

   c. Runs any executable files found in the `.platform/hooks/postdeploy` directory of your source bundle (`.platform/confighooks/postdeploy` for a configuration deployment).

Instance deployment workflow for ECS running on Amazon Linux 2 and later

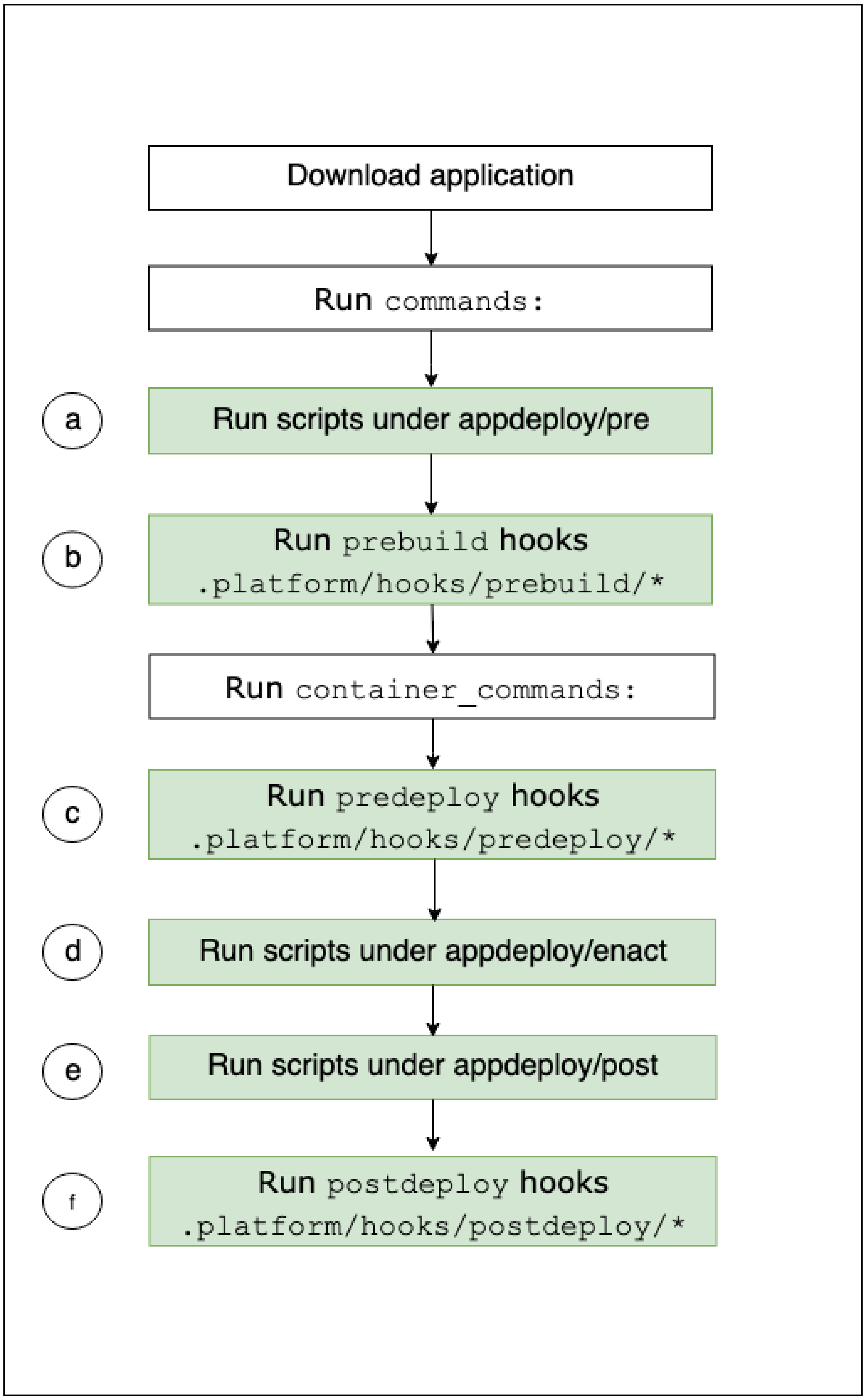
The previous section describes the supported extensibility features throughout the phases of the application deployment workflow. There are some differences for the Docker platform branches *ECS running on Amazon Linux 2 and later*. This section explains how those concepts apply to this specific platform branch.

With many ways to extend your environment's platform, it's useful to know what happens whenever Elastic Beanstalk provisions an instance or runs a deployment to an instance. The following diagram shows this entire deployment workflow for an environment based on the *ECS running on Amazon Linux 2* and *ECS running on Amazon Linux 2023* platform branches. It depicts the different phases in a deployment and the steps that Elastic Beanstalk takes in each phase.

Unlike the workflow described in the prior section, the deployment Configuration phase doesn't support the following extensibility features: `Buildfile` commands, `Procfile` commands, reverse proxy configuration.

Notes

- The diagram doesn't represent the complete set of steps that Elastic Beanstalk takes on environment instances during deployment. We provide this diagram for illustration, to provide you with the order and context for the execution of your customizations.
- For simplicity, the diagram mentions only the `.platform/hooks/*` hook subdirectories (for application deployments), and not the `.platform/confighooks/*` hook subdirectories (for configuration deployments). Hooks in the latter subdirectories run during exactly the same steps as hooks in corresponding subdirectories shown in the diagram.


Download application
Run commands:
a
Run scripts under appdeploy/pre
b
Run prebuild hooks
.platform/hooks/prebuild/*
Run container_commands:
c
Run predeploy hooks
.platform/hooks/predeploy/*
d
Run scripts under appdeploy/enact
e
Run scripts under appdeploy/post
f
Run postdeploy hooks
.platform/hooks/postdeploy/*

The following list details the deployment workflow steps.

a. Runs any executable files found in the `appdeploy/pre` directory under `EBhooksDir`.
b. Runs any executable files found in the `.platform/hooks/prebuild` directory of your source bundle (`.platform/confighooks/prebuild` for a configuration deployment).
c. Runs any executable files found in the `.platform/hooks/predeploy` directory of your source bundle (`.platform/confighooks/predeploy` for a configuration deployment).
d. Runs any executable files found in the `appdeploy/enact` directory under `EBhooksDir`.
e. Runs any executable files found in the `appdeploy/post` directory under `EBhooksDir`.
f. Runs any executable files found in the `.platform/hooks/postdeploy` directory of your source bundle (`.platform/confighooks/postdeploy` for a configuration deployment).

The reference to `EBhooksDir` represents the path of the platform hooks directory. To retrieve directory path name use the get-config script tool on the command line of your environment instance as shown:

```
$ /opt/elasticbeanstalk/bin/get-config platformconfig -k EBhooksDir
```

Platform script tools for your Elastic Beanstalk environments

This topic describes tools that AWS Elastic Beanstalk provides for environments that use Amazon Linux platforms. The tools are located on the Amazon EC2 instances of the Elastic Beanstalk environments.

get-config

Use the `get-config` tool to retrieve plain text environment variable values and other platform and instance information. The tool is available at `/opt/elasticbeanstalk/bin/get-config`.

get-config commands

Each `get-config` tool command returns a specific type of information. Use the following syntax to run the commands of any of the tools.

```
$ /opt/elasticbeanstalk/bin/get-config command [ options ]
```

The following example runs the `environment` command.

```
$ /opt/elasticbeanstalk/bin/get-config environment -k PORT
```

Depending on the command and options you choose, the tool returns an object (JSON or YAML) with key-value pairs or a single value.

You can test `get-config` by using SSH to connect to an EC2 instance in your Elastic Beanstalk environment.

Note

When you run `get-config` for testing, some commands might require root user privileges to access the underlying information. If you get an access permission error, run the command again under `sudo`.
You don't need to add `sudo` when using the tool in the scripts that you deploy to your environment. Elastic Beanstalk runs all your scripts as the root user.

The following sections describe the commands for the tools.

optionsettings – Configuration options

The `get-config optionsettings` command returns an object that's listing the configuration options that are set on the environment and used by the platform on environment instances. They're organized by namespace.

```
$ /opt/elasticbeanstalk/bin/get-config optionsettings
{"aws:elasticbeanstalk:application:environment":
{"JDBC_CONNECTION_STRING":""},"aws:elasticbeanstalk:container:tomcat:jvmoptions":{"JVM
 Options":"","Xms":"256m","Xmx":"256m"},"aws:elasticbeanstalk:environment:proxy":
{"ProxyServer":"nginx","StaticFiles":
[""]},"aws:elasticbeanstalk:healthreporting:system":
{"SystemType":"enhanced"},"aws:elasticbeanstalk:hostmanager":
{"LogPublicationControl":"false"}}
```

To return a specific configuration option value, use the `--namespace` (`-n`) option to specify a namespace, and the `--option-name` (`-o`) option to specify an option name.

```
$ /opt/elasticbeanstalk/bin/get-config optionsettings -
n aws:elasticbeanstalk:container:php:phpini -o memory_limit
256M
```

environment – Environment properties

The `get-config environment` command returns an object containing a list of environment properties, including both user-configured and provided by Elastic Beanstalk. The user-configured properties are defined in the console as *Plain text* or with the configuration option namespace aws:elasticbeanstalk:application:environment.

```
$ /opt/elasticbeanstalk/bin/get-config environment
{"JDBC_CONNECTION_STRING":"","RDS_PORT":"3306","RDS_HOSTNAME":"anj9aw1b0tbj6b.cijbpanmxz5u.us-
west-2.rds.amazonaws.com","RDS_USERNAME":"testusername","RDS_DB_NAME":"ebdb","RDS_PASSWORD":"te
```

For example, Elastic Beanstalk provides environment properties for connecting to an integrated Amazon RDS DB instance (for example, `RDS_HOSTNAME`). These RDS connection properties appear in the output of `get-config environment`. However, they don't appear in the output of `get-config optionsettings`. This is because they weren't set in configuration options.

To return a specific environment property, use the `--key` (`-k`) option to specify a property key.

```
$ /opt/elasticbeanstalk/bin/get-config environment -k TESTPROPERTY
testvalue
```

Note

The `get-config` tool cannot retrieve environment variables that store secrets. For more information about how to programmatically retrieve values from secret or parameter stores, see Using Secrets Manager or Using Systems Manager Parameter Store.

container – On-instance configuration values

The `get-config container` command returns an object that lists platform and environment configuration values for environment instances.

The following example shows the output for the command on an Amazon Linux 2 Tomcat environment.

```
$ /opt/elasticbeanstalk/bin/get-config container
{"common_log_list":["/var/log/eb-engine.log","/var/log/eb-
hooks.log"],"default_log_list":["/var/log/nginx/access.log","/var/log/nginx/
error.log"],"environment_name":"myenv-1da84946","instance_port":"80","log_group_name_prefix":"/
aws/elasticbeanstalk","proxy_server":"nginx","static_files":
[""],"xray_enabled":"false"}
```

To return the value of a specific key, use the `--key` (`-k`) option to specify the key.

```
$ /opt/elasticbeanstalk/bin/get-config container -k environment_name
myenv-1da84946
```

addons – Add-on configuration values

The `get-config addons` command returns an object that contains configuration information of environment add-ons. Use it to retrieve the configuration of an Amazon RDS database that's associated with the environment.

```
$ /opt/elasticbeanstalk/bin/get-config addons
{"rds":{"Description":"RDS Environment variables","env":
{"RDS_DB_NAME":"ebdb","RDS_HOSTNAME":"ea13k2wimu1dh8i.c18mnpu5rwvg.us-
east-2.rds.amazonaws.com","RDS_PASSWORD":"password","RDS_PORT":"3306","RDS_USERNAME":"user"}}}
```

You can restrict the result in two ways. To retrieve values for a specific add-on, use the `--add-on` (`-a`) option to specify the add-on name.

```
$ /opt/elasticbeanstalk/bin/get-config addons -a rds
{"Description":"RDS Environment variables","env":
{"RDS_DB_NAME":"ebdb","RDS_HOSTNAME":"ea13k2wimu1dh8i.c18mnpu5rwvg.us-
east-2.rds.amazonaws.com","RDS_PASSWORD":"password","RDS_PORT":"3306","RDS_USERNAME":"user"}}
```

To return the value of a specific key within an add-on, add the `--key` (`-k`) option to specify the key.

```
$ /opt/elasticbeanstalk/bin/get-config addons -a rds -k RDS_DB_NAME
ebdb
```

platformconfig – Constant configuration values

The `get-config platformconfig` command returns an object that contains platform configuration information that's constant to the platform version. The output is the same on all environments that run the same platform version. The output object for the command has two embedded objects:

- `GeneralConfig` – Contains information that's constant across the latest versions of all Amazon Linux 2 and Amazon Linux 2023 platform branches.
- `PlatformSpecificConfig` – Contains information that's constant for the platform version and is specific to it.

The following example shows the output for the command on an environment that uses the *Tomcat 8.5 running Corretto 11* platform branch.

```
$ /opt/elasticbeanstalk/bin/get-config platformconfig
{"GeneralConfig":{"AppUser":"webapp","AppDeployDir":"/var/app/
current/","AppStagingDir":"/var/app/
staging/","ProxyServer":"nginx","DefaultInstancePort":"80"},"PlatformSpecificConfig":
{"ApplicationPort":"8080","JavaVersion":"11","TomcatVersion":"8.5"}}
```

To return the value of a specific key, use the `--key` (`-k`) option to specify the key. These keys are unique across the two embedded objects. You don't need to specify the object that contains the key.

```
$ /opt/elasticbeanstalk/bin/get-config platformconfig -k AppStagingDir
/var/app/staging/
```

get-config output options

Use the `--output` option to specify the output object format. Valid values are `JSON` (default) and `YAML`. This is a global option. You must specify it before the command name.

The following example returns configuration option values in the YAML format.

```
$ /opt/elasticbeanstalk/bin/get-config --output YAML optionsettings
aws:elasticbeanstalk:application:environment:
  JDBC_CONNECTION_STRING: ""
aws:elasticbeanstalk:container:tomcat:jvmoptions:
  JVM Options: ""
  Xms: 256m
  Xmx: 256m
aws:elasticbeanstalk:environment:proxy:
  ProxyServer: nginx
  StaticFiles:
        - ""
aws:elasticbeanstalk:healthreporting:system:
  SystemType: enhanced
aws:elasticbeanstalk:hostmanager:
  LogPublicationControl: "false"
```

pkg-repo

Note

The pkg-repo tool is not available for environments based on Amazon Linux 2023 platforms. However, you can manually apply package and operating system updates to an AL2023 instance. For more information, see Managing packages and operating system updates in the *Amazon Linux 2023 User Guide*

In some urgent circumstances, you might need to update your Amazon EC2 instances with an Amazon Linux 2 security patch that hasn't yet been released with the required Elastic Beanstalk platform versions. You can't perform a manual update on your Elastic Beanstalk environments by default. This is because the platform versions are locked to a specific version of the Amazon Linux 2 repository. This lock ensures that instances run supported and consistent software versions. For urgent cases, the pkg-repo tool allows a workaround to manually update yum packages on Amazon Linux 2 if you need to install it on an environment before it's released in a new Elastic Beanstalk platform version.

The pkg-repo tool on Amazon Linux 2 platforms provides the capability to unlock the yum package repositories. You can then manually perform a **yum update** for a security patch. Conversely, you can follow the update by using the tool to lock the yum package repositories to prevent further updates. The pkg-repo tool is available at the /opt/elasticbeanstalk/bin/pkg-repo directory of all the EC2 instances in your Elastic Beanstalk environments.

Changes using the pkg-repo tool are made only on the EC2 instance that the tool is used on. They don’t affect other instances or prevent future updates to the environment. The examples that are provided later in this topic explain how to apply the changes across all instances by calling the pkg-repo commands from scripts and configuration files.

Warning

We don't recommend this tool for most users. Any manual changes applied to an unlocked platform version are considered out of band. This option is only viable for those users in urgent circumstances that can accept the following risks:

- Package versions can't be guaranteed to be consistent across all instances in your environments.
- Environments that are modified using the `pkg-repo` tool aren't guaranteed to function properly. They haven't been tested and verified on Elastic Beanstalk supported platforms.

We strongly recommend applying best practices that include testing and backout plans. To help facilitate best practices, you can use the Elastic Beanstalk console and EB CLI to clone an environment and swap environment URLs. For more information about using these operations, see Blue/Green deployments in the *Managing environments* chapter of this guide.

If you plan to manually edit yum repository configuration files, run the `pkg-repo` tool first. The `pkg-repo` tool might not work as intended in an Amazon Linux 2 environment with manually edited yum repository configuration files. This is because the tool might not recognize the configuration changes.

For more information about the Amazon Linux package repository, see the Package repository topic in the *Amazon EC2 User Guide*.

pkg-repo commands

Use the following syntax to run the `pkg-repo` tool commands.

```
$ /opt/elasticbeanstalk/bin/pkg-repo command [options]
```

The `pkg-repo` commands are the following:

- **lock** – locks the `yum` package repositories to a specific version
- **unlock** – unlocks the `yum` package repositories from a specific version
- **status** – lists all the `yum` package repositories and their current lock status
- **help** – shows general help or help for one command

The options apply to the commands as follows:

- `lock`, `unlock` and `status` – options: `-h`, `--help`, or none (default).

- `help` – options: `lock`, `unlock`, `status`, or none (default).

The following example runs the **unlock** command.

```
$ sudo /opt/elasticbeanstalk/bin/pkg-repo unlock
Amazon Linux 2 core package repo successfully unlocked
Amazon Linux 2 extras package repo successfully unlocked
```

The following example runs the **lock** command.

```
$ sudo /opt/elasticbeanstalk/bin/pkg-repo lock
Amazon Linux 2 core package repo successfully locked
Amazon Linux 2 extras package repo successfully locked
```

The following example runs the **status** command.

```
$ sudo /opt/elasticbeanstalk/bin/pkg-repo status
Amazon Linux 2 core package repo is currently UNLOCKED
Amazon Linux 2 extras package repo is currently UNLOCKED
```

The following example runs the **help** command for the **lock** command.

```
$ sudo /opt/elasticbeanstalk/bin/pkg-repo help lock
```

The following example runs the **help** command for the `pkg-repo` tool.

```
$ sudo /opt/elasticbeanstalk/bin/pkg-repo help
```

You can test `pkg-repo` by using SSH to connect to an instance in your Elastic Beanstalk environment. One SSH option is the EB CLI **eb ssh** command.

Note

The `pkg-repo` tool requires root user privileges to run. If you get an access permission error, run the command again under `sudo`.
You don't need to add `sudo` when using the tool in the scripts or configuration files that you deploy to your environment. Elastic Beanstalk runs all your scripts as the root user.

pkg-repo examples

The previous section provides command line examples for testing on an individual EC2 instance of an Elastic Beanstalk environment. This approach can be helpful for testing. However, it updates only one instance at a time, so it isn't practical for applying changes to all of the instances in an environment.

A more pragmatic approach is to use platform hook scripts or an `.ebextensions` configuration file to apply the changes across all instances in a consistent manner.

The following example calls `pkg-repo` from a configuration file in the `.ebextensions` folder. Elastic Beanstalk runs the commands in the `update_package.config` file when you deploy your application source bundle.

```
.ebextensions
### update_package.config
```

To receive the latest version of the *docker* package, this configuration specifies the *docker* package in the **yum update** command.

```
### update_package.config ###

commands:
  update_package:
    command: |
      /opt/elasticbeanstalk/bin/pkg-repo unlock
      yum update docker -y
      /opt/elasticbeanstalk/bin/pkg-repo lock
      yum clean all -y
      rm -rf /var/cache/yum
```

This configuration doesn't specify any packages in the **yum update** command. All available updates are applied as a result.

```
### update_package.config ###

commands:
  update_package:
    command: |
      /opt/elasticbeanstalk/bin/pkg-repo unlock
```

```
        yum update -y
        /opt/elasticbeanstalk/bin/pkg-repo lock
        yum clean all -y
        rm -rf /var/cache/yum
```

The following example calls `pkg-repo` from a bash script as a platform hook. Elastic Beanstalk runs the `update_package.sh` script file that's located in the `prebuild` subdirectory.

```
.platform
### hooks
    ### prebuild
        ### update_package.sh
```

To receive the latest version of the *docker* package, this script specifies the *docker* package in the **yum update** command. If the package name is omitted, all the available updates are applied. The prior configuration file example demonstrates this.

```
### update_package.sh ###

#!/bin/bash

/opt/elasticbeanstalk/bin/pkg-repo unlock
yum update docker -y
/opt/elasticbeanstalk/bin/pkg-repo lock
yum clean all -y
rm -rf /var/cache/yum
```

download-source-bundle (Amazon Linux AMI only)

On Amazon Linux AMI platform branches (preceding Amazon Linux 2), Elastic Beanstalk provides an additional tool, which is `download-source-bundle`. Use this tool to download your application source code when deploying your platform. The tool is available at `/opt/elasticbeanstalk/bin/download-source-bundle`.

The example script `00-unzip.sh` is located in the `appdeploy/pre` folder on environment instances. It demonstrates how to use `download-source-bundle` to download the application source code to the `/opt/elasticbeanstalk/deploy/appsource` folder during deployment.

Extending Elastic Beanstalk Linux platforms

This topic describes how to extend your Linux platforms with your own commands, scripts, software, and configurations. You may need to extend your platform to change the default proxy server and configuration. Or you may need to customize how the platform builds or starts up your application.

Topics

Buildfile and Procfile

Some platforms allow you to customize how you build or prepare your application, and to specify the processes that run your application. Each individual platform topic specifically mentions *Buildfile* and/or *Procfile* if the platform supports them. Look for your specific platform under *Platforms*.

For all supporting platforms, syntax and semantics are identical, and are as described on this page. Individual platform topics mention specific usage of these files for building and running applications in their respective languages.

Buildfile

To specify a custom build and configuration command for your application, place a file named `Buildfile` in the root directory of your application source. The file name is case sensitive. Use the following syntax for your `Buildfile`.

```
<process_name>: <command>
```

The command in your `Buildfile` must match the following regular expression: `^[A-Za-z0-9_-]+:\s*[^\s].*$`

Elastic Beanstalk doesn't monitor the application that is run with a `Buildfile`. Use a `Buildfile` for commands that run for short periods and terminate after completing their tasks. For long-running application processes that should not exit, use a Procfile.

All paths in the `Buildfile` are relative to the root of the source bundle. In the following example of a `Buildfile`, `build.sh` is a shell script located at the root of the source bundle.

Example Buildfile

```
make: ./build.sh
```

If you want to provide custom build steps, we recommend that you use `predeploy` platform hooks for anything but the simplest commands, instead of a `Buildfile`. Platform hooks allow richer scripts and better error handling. Platform hooks are described in the next section.

Procfile

To specify custom commands to start and run your application, place a file named `Procfile` in the root directory of your application source. The file name is case sensitive. Use the following syntax for your `Procfile`. You can specify one or more commands.

```
<process_name1>: <command1>
<process_name2>: <command2>
...
```

Each line in your `Procfile` must match the following regular expression: `^[A-Za-z0-9_-]+:\s*[^\s].*$`

Use a `Procfile` for long-running application processes that shouldn't exit. Elastic Beanstalk expects processes run from the `Procfile` to run continuously. Elastic Beanstalk monitors these processes and restarts any process that terminates. For short-running processes, use a Buildfile.

All paths in the `Procfile` are relative to the root of the source bundle. The following example `Procfile` defines three processes. The first one, called `web` in the example, is the *main web application*.

Example Procfile

```
web: bin/myserver
cache: bin/mycache
```

```
foo: bin/fooapp
```

Elastic Beanstalk configures the proxy server to forward requests to your main web application on port 5000, and you can configure this port number. A common use for a `Procfile` is to pass this port number to your application as a command argument. For details about proxy configuration, see Reverse proxy configuration.

Elastic Beanstalk captures standard output and error streams from `Procfile` processes in log files. Elastic Beanstalk names the log files after the process and stores them in `/var/log`. For example, the `web` process in the preceding example generates logs named `web-1.log` and `web-1.error.log` for `stdout` and `stderr`, respectively.

Platform hooks

Platform hooks are specifically designed to extend your environment's platform. These are custom scripts and other executable files that you deploy as part of your application's source code, and Elastic Beanstalk runs during various instance provisioning stages.

Note

Platform hooks aren't supported on Amazon Linux AMI platform versions (preceding Amazon Linux 2).

Application deployment platform hooks

An *application deployment* occurs when you provide a new source bundle for deployment, or when you make a configuration change that requires termination and recreation of all environment instances.

To provide platform hooks that run during an application deployment, place the files under the `.platform/hooks` directory in your source bundle, in one of the following subdirectories.

- `prebuild` – Files here run after the Elastic Beanstalk platform engine downloads and extracts the application source bundle, and before it sets up and configures the application and web server.

  The `prebuild` files run after running commands found in the commands section of any configuration file and before running `Buildfile` commands.

- `predeploy` – Files here run after the Elastic Beanstalk platform engine sets up and configures the application and web server, and before it deploys them to their final runtime location.

  The `predeploy` files run after running commands found in the container_commands section of any configuration file and before running `Procfile` commands.
- `postdeploy` – Files here run after the Elastic Beanstalk platform engine deploys the application and proxy server.

  This is the last deployment workflow step.

Configuration deployment platform hooks

A *configuration deployment* occurs when you make configuration changes that only update environment instances without recreating them. The following option updates cause a configuration update.

- Environment properties and platform-specific settings
- Static files
- AWS X-Ray daemon
- Log storage and streaming
- Application port (for details see Reverse proxy configuration)

To provide hooks that run during a configuration deployment, place them under the `.platform/confighooks` directory in your source bundle. The same three subdirectories as for application deployment hooks apply.

More about platform hooks

Hook files can be binary files, or script files starting with a `#!` line containing their interpreter path, such as `#!/bin/bash`. All files must have execute permission. Use `chmod +x` to set execute permission on your hook files. For all Amazon Linux 2023 and Amazon Linux 2 based platforms versions that were released on or after April 29, 2022, Elastic Beanstalk automatically grants execute permissions to all of the platform hook scripts. In this case you don't have to manually grant execute permissions. For a list of these platform versions, refer to the April 29, 2022 Linux release notes in the *AWS Elastic Beanstalk Release Notes Guide*.

Elastic Beanstalk runs files in each one of these directories in lexicographical order of file names. All files run as the `root` user. The current working directory (cwd) for platform hooks is the application's root directory. For `prebuild` and `predeploy` files it's the application staging directory, and for `postdeploy` files it's the current application directory. If one of the files fails (exits with a non-zero exit code), the deployment aborts and fails.

A platform hooks text script may fail if it contains Windows *Carriage Return / Line Feed* (CRLF) line break characters. If a file was saved in a Windows host, then transferred to a Linux server, it may contain Windows CRLF line breaks. For platforms released on or after December 29, 2022, Elastic Beanstalk automatically converts Windows CRLF characters to Linux *Line Feed* (LF) line break characters in platform hooks text files. If you application runs on any Amazon Linux 2 platforms that were release prior to this date, you'll need to convert the Windows CRLF characters to Linux LF characters. One way to accomplish this is to create and save the script file on a Linux host. Tools that convert these characters are also available on the internet.

Hook files have access to all environment properties that you've defined in application options, and to the system environment variables `HOME`, `PATH`, and `PORT`.

To get values of environment variables and other configuration options into your platform hook scripts, you can use the `get-config` utility that Elastic Beanstalk provides on environment instances. For details, see the section called “Platform script tools”.

Configuration files

You can add configuration files to the `.ebextensions` directory of your application's source code to configure various aspects of your Elastic Beanstalk environment. Among other things, configuration files let you customize software and other files on your environment's instances and run initialization commands on the instances. For more information, see the section called “Linux server”.

You can also set configuration options using configuration files. Many of the options control platform behavior, and some of these options are platform specific.

For platforms based on Amazon Linux 2 and Amazon Linux 2023, we recommend using *Buildfile*, *Procfile*, and *platform hooks* to configure and run custom code on your environment instances during instance provisioning. These mechanisms are described in the previous sections on this page. You can still use commands and container commands in `.ebextensions` configuration files, but they aren't as easy to work with. For example, writing command scripts inside a YAML file can be challenging from a syntax standpoint. You still need to use `.ebextensions` configuration files for any script that needs a reference to a AWS CloudFormation resource.

Reverse proxy configuration

All Amazon Linux 2 and Amazon Linux 2023 platform versions use nginx as their default reverse proxy server. The Tomcat, Node.js, PHP, and Python platform also support Apache HTTPD as an alternative. To select Apache on these platforms, set the `ProxyServer` option in the `aws:elasticbeanstalk:environment:proxy` namespace to `apache`. All platforms enable proxy server configuration in a uniform way, as described in this section.

Note

On Amazon Linux AMI platform versions (preceding Amazon Linux 2) you might have to configure proxy servers differently. You can find these legacy details under the respective platform topics in this guide.

Elastic Beanstalk configures the proxy server on your environment's instances to forward web traffic to the main web application on the root URL of the environment; for example, `http://my-env.elasticbeanstalk.com`.

By default, Elastic Beanstalk configures the proxy to forward requests coming in on port 80 to your main web application on port 5000. You can configure this port number by setting the `PORT` environment property using the aws:elasticbeanstalk:application:environment namespace in a configuration file, as shown in the following example.

```
option_settings:
  - namespace:  aws:elasticbeanstalk:application:environment
    option_name:  PORT
    value:  <main_port_number>
```

For more information about setting environment variables for your application, see the section called “Option settings”.

Your application should listen on the port that is configured for it in the proxy. If you change the default port using the `PORT` environment property, your code can access it by reading the value of the `PORT` environment variable. For example, call `os.Getenv("PORT")` in Go, or `System.getenv("PORT")` in Java. If you configure your proxy to send traffic to multiple application processes, you can configure several environment properties, and use their values in both proxy configuration and your application code. Another option is to pass the port value to the process as a command argument in the `Procfile`. For more information see Buildfile and Procfile.

Configuring nginx

Elastic Beanstalk uses nginx as the default reverse proxy to map your application to your Elastic Load Balancing load balancer. Elastic Beanstalk provides a default nginx configuration that you can extend or override completely with your own configuration.

Note

When you add or edit an nginx `.conf` configuration file, be sure to encode it as UTF-8.

To extend the Elastic Beanstalk default nginx configuration, add `.conf` configuration files to a folder named `.platform/nginx/conf.d/` in your application source bundle. The Elastic Beanstalk nginx configuration includes `.conf` files in this folder automatically.

```
~/workspace/my-app/
|-- .platform
|   `-- nginx
|       `-- conf.d
|           `-- myconf.conf
`-- other source files
```

To override the Elastic Beanstalk default nginx configuration completely, include a configuration in your source bundle at `.platform/nginx/nginx.conf`:

```
~/workspace/my-app/
|-- .platform
|   `-- nginx
|       `-- nginx.conf
`-- other source files
```

If you override the Elastic Beanstalk nginx configuration, add the following line to your `nginx.conf` to pull in the Elastic Beanstalk configurations for Enhanced health reporting and monitoring in Elastic Beanstalk, automatic application mappings, and static files.

```
include conf.d/elasticbeanstalk/*.conf;
```

Configuring Apache HTTPD

The Tomcat, Node.js, PHP, and Python platforms allow you to choose the Apache HTTPD proxy server as an alternative to nginx. This isn't the default. The following example configures Elastic Beanstalk to use Apache HTTPD.

Example .ebextensions/httpd-proxy.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: apache
```

You can extend the Elastic Beanstalk default Apache configuration with your additional configuration files. Alternatively, you can override the Elastic Beanstalk default Apache configuration completely.

To extend the Elastic Beanstalk default Apache configuration, add `.conf` configuration files to a folder named `.platform/httpd/conf.d` in your application source bundle. The Elastic Beanstalk Apache configuration includes `.conf` files in this folder automatically.

```
~/workspace/my-app/
|-- .ebextensions
|   -- httpd-proxy.config
|-- .platform
|   -- httpd
|      -- conf.d
|         -- port5000.conf
|         -- ssl.conf
-- index.jsp
```

For example, the following Apache 2.4 configuration adds a listener on port 5000.

Example .platform/httpd/conf.d/port5000.conf

```
listen 5000
<VirtualHost *:5000>
  <Proxy *>
    Require all granted
  </Proxy>
  ProxyPass / http://localhost:8080/ retry=0
  ProxyPassReverse / http://localhost:8080/
  ProxyPreserveHost on

  ErrorLog /var/log/httpd/elasticbeanstalk-error_log
</VirtualHost>
```

To override the Elastic Beanstalk default Apache configuration completely, include a configuration in your source bundle at `.platform/httpd/conf/httpd.conf`.

```
~/workspace/my-app/
|-- .ebextensions
|   -- httpd-proxy.config
|-- .platform
|   `-- httpd
|       `-- conf
|           `-- httpd.conf
`-- index.jsp
```

Note

If you override the Elastic Beanstalk Apache configuration, add the following lines to your `httpd.conf` to pull in the Elastic Beanstalk configurations for Enhanced health reporting and monitoring in Elastic Beanstalk, automatic application mappings, and static files.

```
IncludeOptional conf.d/elasticbeanstalk/*.conf
```

Application example with extensions

The following example demonstrates an application source bundle with several extensibility features that Elastic Beanstalk Amazon Linux 2 and Amazon Linux 2023 platforms support: a `Procfile`, `.ebextensions` configuration files, custom hooks, and proxy configuration files.

```
~/my-app/
|-- web.jar
|-- Procfile
|-- readme.md
|-- .ebextensions/
```

```
|   |-- options.config        # Option settings
|   `-- cloudwatch.config     # Other .ebextensions sections, for example files and
 container commands
`-- .platform/
    |-- nginx/                # Proxy configuration
    |   |-- nginx.conf
    |   `-- conf.d/
    |       `-- custom.conf
    |-- hooks/                # Application deployment hooks
    |   |-- prebuild/
    |   |   |-- 01_set_secrets.sh
    |   |   `-- 12_update_permissions.sh
    |   |-- predeploy/
    |   |   `-- 01_some_service_stop.sh
    |   `-- postdeploy/
    |       |-- 01_set_tmp_file_permissions.sh
    |       |-- 50_run_something_after_app_deployment.sh
    |       `-- 99_some_service_start.sh
    `-- confighooks/          # Configuration deployment hooks
        |-- prebuild/
        |   `-- 01_set_secrets.sh
        |-- predeploy/
        |   `-- 01_some_service_stop.sh
        `-- postdeploy/
            |-- 01_run_something_after_config_deployment.sh
            `-- 99_some_service_start.sh
```

Note

Some of these extensions aren't supported on Amazon Linux AMI platform versions (preceding Amazon Linux 2).

Deploying .NET Windows applications with Elastic Beanstalk

Check out the *.NET on AWS Developer Center*

Have you stopped by our *.Net Developer Center*? It's our one stop shop for all things .NET on AWS.

For more information see the .NET on AWS Developer Center.

This chapter provides instructions for configuring and deploying your ASP.NET and .NET Core Windows web applications to AWS Elastic Beanstalk. Elastic Beanstalk makes it easy to deploy, manage, and scale your .NET (Windows) web applications using Amazon Web Services.

You can deploy your application in just a few minutes using the Elastic Beanstalk Command Line Interface (EB CLI) or by using the Elastic Beanstalk console. After you deploy your Elastic Beanstalk application, you can continue to use the EB CLI to manage your application and environment, or you can use the Elastic Beanstalk console, AWS CLI, or the APIs.

This chapter provides the following tutorials:

- QuickStart for .NET Core on Windows — Step-by-step instructions to create and deploy a *Hello World* .NET Core Windows application using the EB CLI.
- QuickStart for ASP.NET — Step-by-step instructions to create and deploy a *Hello World* ASP.NET application using the AWS Toolkit for Visual Studio.

If you need help with Windows .NET Core application development, there are several places you can go:

- .NET Development Forum — Post your questions and get feedback.
- .NET Developer Center — One-stop shop for sample code, documentation, tools, and additional resources.
- AWS SDK for .NET Documentation — Read about setting up the SDK and running code samples, features of the SDK, and detailed information about the API operations for the SDK.

Note

This platform does not support worker environments. For details, see Elastic Beanstalk worker environments.

Topics

QuickStart: Deploy a .NET Core on Windows application to Elastic Beanstalk

This QuickStart tutorial walks you through the process of creating a .NET Core on Windows application and deploying it to an AWS Elastic Beanstalk environment.

Note

Tutorial examples are intended for demonstration. Do not use the application for production traffic.

Sections

Your AWS account

If you're not already an AWS customer, you need to create an AWS account. Signing up enables you to access Elastic Beanstalk and other AWS services that you need.

If you already have an AWS account, you can move on to Prerequisites.

Create an AWS account

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.
2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.
2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Prerequisites

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol (>) and the name of the current directory, when appropriate.

```
C:\eb-project> this is a command
this is output
```

EB CLI

This tutorial uses the Elastic Beanstalk Command Line Interface (EB CLI). For details on installing and configuring the EB CLI, see Install EB CLI with setup script (recommended) and Configure the EB CLI.

.NET Core on Windows

If you don't have the .NET SDK installed on your local machine, you can install it by following the Download .NET link on the .NET documentation website.

Verify your .NET SDK installation by running the following command.

```
C:\> dotnet --info
```

Step 1: Create a .NET Core on Windows application

Create a project directory.

```
C:\> mkdir eb-dotnetcore
```

```
C:\> cd eb-dotnetcore
```

Next, create a sample Hello World RESTful web service application by running the following commands.

```
C:\eb-dotnetcore> dotnet new web --name HelloElasticBeanstalk
C:\eb-dotnetcore> cd HelloElasticBeanstalk
```

Step 2: Run your application locally

Run the following command to run your application locally.

```
C:\eb-dotnetcore\HelloElasticBeasntalk> dotnet run
```

The output should look something like the following text.

```
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7222
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5228
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\Administrator\eb-dotnetcore\HelloElasticBeanstalk
```

Note

The `dotnet` command selects a port at random when running the application locally. In this example the port is 5228. When you deploy the application to your Elastic Beanstalk environment, the application will run on port 5000.

Enter the URL address `http://localhost:`*`port`* in your web browser. For this specific example, the command is `http://localhost:5228`. The web browser should display “Hello World!”.

Step 3: Deploy your .NET Core on Windows application with the EB CLI

Run the following commands to create an Elastic Beanstalk environment for this application.

To create an environment and deploy your .NET Core on Windows application

1. Run the following commands in the `HelloElasticBeanstalk` directory to publish and zip your application.

   ```
   C:\eb-dotnetcore\HelloElasticBeasntalk> dotnet publish -o site
   C:\eb-dotnetcore\HelloElasticBeasntalk> cd site
   C:\eb-dotnetcore\HelloElasticBeasntalk\site> Compress-Archive -Path * -
   DestinationPath ../site.zip
   C:\eb-dotnetcore\HelloElasticBeasntalk\site> cd ..
   ```

2. Create a new file in the `HelloElasticBeanstalk` called `aws-windows-deployment-manifest.json` with the following contents:

   ```
   {
       "manifestVersion": 1,
       "deployments": {
           "aspNetCoreWeb": [
           {
               "name": "test-dotnet-core",
               "parameters": {
                   "appBundle": "site.zip",
                   "iisPath": "/",
                   "iisWebSite": "Default Web Site"
               }
           }
           ]
       }
   }
   ```

3. Initialize your EB CLI repository with the **eb init** command.

   ```
   C:\eb-dotnetcore\HelloElasticBeasntalk> eb init -p iis dotnet-windows-server-
   tutorial --region us-east-2
   ```

   This command creates an application named `dotnet-windows-server-tutorial` and configures your local repository to create environments with the latest Windows server platform version.

4. Create an environment and deploy your application to it with **eb create**. Elastic Beanstalk automatically builds a zip file for your application and starts it on port 5000.

```
C:\eb-dotnetcore\HelloElasticBeasntalk> eb create dotnet-windows-server-env
```

It takes about five minutes for Elastic Beanstalk to create your environment.

Step 4: Run your application on Elastic Beanstalk

When the process to create your environment completes, open your website with **eb open**.

```
C:\eb-dotnetcore\HelloElasticBeasntalk> eb open
```

Congratulations! You've deployed a .NET Core on Windows application with Elastic Beanstalk! This opens a browser window using the domain name created for your application.

Step 5: Clean up

You can terminate your environment when you finish working with your application. Elastic Beanstalk terminates all AWS resources associated with your environment.

To terminate your Elastic Beanstalk environment with the EB CLI run the following command.

```
C:\eb-dotnetcore\HelloElasticBeasntalk> eb terminate
```

AWS resources for your application

You just created a single instance application. It serves as a straightforward sample application with a single EC2 instance, so it doesn't require load balancing or auto scaling. For single instance applications Elastic Beanstalk creates the following AWS resources:

- **EC2 instance** – An Amazon EC2 virtual machine configured to run web apps on the platform you choose.

  Each platform runs a different set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination thereof. Most platforms use either Apache or nginx as a reverse proxy that processes web traffic in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group** – An Amazon EC2 security group configured to allow incoming traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic is not allowed on other ports.

- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name** – A domain name that routes to your web app in the form *`subdomain`*.*`region`*.*elasticbeanstalk.com*.

Elastic Beanstalk manages all of these resources. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Next steps

After you have an environment running an application, you can deploy a new version of the application or a different application at any time. Deploying a new application version is very quick because it doesn't require provisioning or restarting EC2 instances. You can also explore your new environment using the Elastic Beanstalk console. For detailed steps, see Explore your environment in the *Getting started* chapter of this guide.

Try more tutorials

If you'd like to try other tutorials with different example applications, see QuickStart for ASP.NET.

After you deploy a sample application or two and are ready to start developing and running .NET Core on Windows applications locally, see Setting up your .NET development environment

Deploy with the Elastic Beanstalk console

You can also use the Elastic Beanstalk console to launch the sample application. For detailed steps, see Create an example application in the *Getting started* chapter of this guide.

QuickStart: Deploy an ASP.NET application to Elastic Beanstalk

This QuickStart tutorial walks you through the process of creating a ASP.NET application and deploying it to an AWS Elastic Beanstalk environment.

Note

Tutorial examples are intended for demonstration. Do not use the application for production traffic.

Sections

Your AWS account

If you're not already an AWS customer, you need to create an AWS account. Signing up enables you to access Elastic Beanstalk and other AWS services that you need.

If you already have an AWS account, you can move on to Prerequisites.

Create an AWS account

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.
2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Prerequisites

This QuickStart tutorial walks you through creating a "Hello World" application and deploying it to an Elastic Beanstalk environment with Visual Studio and the AWS Toolkit for Visual Studio.

Visual Studio

To download and install Visual Studio follow the instructions on the Visual Studio download page. This example uses Visual Studio 2022. During the Visual Studio installation select these specific items:

- On the **Workloads** tab — select **ASP.NET and web development**.

- On the **Individual components** tab — select **.NET Framework 4.8 development tools** and **.NET Framework project and item templates**.

AWS Toolkit for Visual Studio

To download and set up AWS Toolkit for Visual Studio follow the instructions in the Getting started topic of the AWS Toolkit for Visual Studio User Guide.

Step 1: Create a ASP.NET application

Next, create an application that you'll deploy to an Elastic Beanstalk environment. We'll create a "Hello World" ASP.NET web application.

To create an ASP.NET application

1. Launch Visual Studio. In the **File** menu, select **New**, then **Project**.
2. The **Create a new project** dialog box displays. Select **ASP.NET web application (.NET Framework)**, then select **Next**.
3. On the **Configure your new project** dialog, enter `eb-aspnet` for your **Project name**. From the **Framework** dropdown menu select **.NET Framework 4.8**, then select **Create**.

   Note the project directory. In this example, the project directory is `C:\Users\Administrator\source\repos\eb-aspnet\eb-aspnet`.
4. The **Create a new ASP.NET Web Application** dialogue displays. Select the **Empty** template. Next select **Create**.

   At this point, you have created an empty ASP.NET web application project using Visual Studio. Next, we'll create a web form that will serve as the entry point for the ASP.NET web application.
5. From the **Project** menu, select **Add New Item**. On the **Add New Item** page, select **Web Form** and name it `Default.aspx`. Next select **Add**.
6. Add the following to `Default.aspx:`

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
 Inherits="eb_aspnet.Default" %>

<!DOCTYPE html>
```

```
<html xmlns="https://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Hello Elastic Beanstalk!</title>
</head>
<body>
    <form id="body" runat="server">
        <div>
            Hello Elastic Beanstalk! This is an ASP.NET on Windows Server
 application.
        </div>
    </form>
</body>
</html>
```

Step 2: Run your application locally

In Visual Studio, from the **Debug** menu select **Start Debugging** to run your application locally. The page should display "Hello Elastic Beanstalk! This is an ASP.NET on Windows Server application."

Step 3: Deploy your ASP.NET application with the AWS Toolkit for Visual Studio

Follow these steps to create an Elastic Beanstalk environment and deploy your new application to it.

To create an environment and deploy your ASP.NET application

1. In **Solution Explorer**, right-click your application, then select **Publish to AWS Elastic Beanstalk**.
2. Choose a name for your new Elastic Beanstalk application and environment.
3. Beyond this point, you may proceed with the defaults provided by Elastic Beanstalk or modify any of the options and settings to your liking.
4. On the **Review** page, select **Deploy**. This will package your ASP.NET web application and deploy it to Elastic Beanstalk.

   It takes about five minutes for Elastic Beanstalk to create your environment. The Elastic Beanstalk deployment feature will monitor your environment until it becomes available with the newly deployed code. On the **Env:`<environment name>`** tab, you'll see the status for your environment.

Step 4: Run your application on Elastic Beanstalk

When the process to create your environment completes, the **`Env:<environment name>`** tab, displays information about your environment and application, including the domain URL to launch your application. Select this URL on this tab or copy and paste it into your web browser.

Congratulations! You've deployed a ASP.NET application with Elastic Beanstalk!

Step 5: Clean up

When you finish working with your application, you can terminate your environment in the AWS Toolkit for Visual Studio.

To terminate your environment

1. Expand the Elastic Beanstalk node and the application node in **AWS Explorer**. Right-click your application environment and select **Terminate Environment**.
2. When prompted, select **Yes** to confirm that you want to terminate the environment. It will take a few minutes for Elastic Beanstalk to terminate the AWS resources running in the environment.

AWS resources for your application

You just created a single instance application. It serves as a straightforward sample application with a single EC2 instance, so it doesn't require load balancing or auto scaling. For single instance applications Elastic Beanstalk creates the following AWS resources:

- **EC2 instance** – An Amazon EC2 virtual machine configured to run web apps on the platform you choose.

  Each platform runs a different set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination thereof. Most platforms use either Apache or nginx as a reverse proxy that processes web traffic in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group –** An Amazon EC2 security group configured to allow incoming traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic is not allowed on other ports.

- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name** – A domain name that routes to your web app in the form *subdomain*.*region*.*elasticbeanstalk.com*.

Elastic Beanstalk manages all of these resources. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Next steps

After you have an environment running an application, you can deploy a new version of the application or a different application at any time. Deploying a new application version is very quick because it doesn't require provisioning or restarting EC2 instances. You can also explore your new environment using the Elastic Beanstalk console. For detailed steps, see Explore your environment in the *Getting started* chapter of this guide.

Try more tutorials

If you'd like to try other tutorials with different example applications, see QuickStart for .NET Core on Windows.

After you deploy a sample application or two and are ready to start developing and running ASP.NET applications locally, see Setting up your .NET development environment

Deploy with the Elastic Beanstalk console

You can also use the Elastic Beanstalk console to launch the sample application. For detailed steps, see Create an example application in the *Getting started* chapter of this guide.

Setting up your .NET development environment

This topic provides instructions to set up a .NET Windows development environment to test your application locally prior to deploying it to AWS Elastic Beanstalk. It also references websites that provide installation instructions for useful tools.

Sections

If you need to manage AWS resources from within your application, install the AWS SDK for .NET. For example, you can use Amazon S3 to store and retrieve data.

With the AWS SDK for .NET, you can get started in minutes with a single, downloadable package complete with Visual Studio project templates, the AWS .NET library, C# code samples, and documentation. Practical examples are provided in C# for how to use the libraries to build applications. Online video tutorials and reference documentation are provided to help you learn how to use the libraries and code samples.

Visit the AWS SDK for .NET homepage for more information and installation instructions.

Installing an IDE

Integrated development environments (IDEs) provide a wide range of features that facilitate application development. If you haven't used an IDE for .NET development, try Visual Studio Community to get started.

Visit the Visual Studio Community page to download and install Visual Studio Community.

Installing the AWS Toolkit for Visual Studio

The AWS Toolkit for Visual Studio is an open source plug-in for the Visual Studio IDE that makes it easier for developers to develop, debug, and deploy .NET applications using AWS. Visit the Toolkit for Visual Studio homepage for installation instructions.

Using the Elastic Beanstalk .NET Windows platform

This topic describes how to configure, build, and run your ASP.NET and .NET Core Windows web applications on Elastic Beanstalk.

AWS Elastic Beanstalk supports a number of platforms for different versions of the .NET programming framework and Windows Server. See .NET on Windows Server with IIS in the *AWS Elastic Beanstalk Platforms* document for a full list.

Elastic Beanstalk provides configuration options that you can use to customize the software that runs on the EC2 instances in your Elastic Beanstalk environment. You can configure environment variables needed by your application, enable log rotation to Amazon S3, and set .NET framework settings.

Configuration options are available in the Elastic Beanstalk console for modifying the configuration of a running environment. To avoid losing your environment's configuration when you terminate it, you can use saved configurations to save your settings and later apply them to another environment.

To save settings in your source code, you can include configuration files. Settings in configuration files are applied every time you create an environment or deploy your application. You can also use configuration files to install packages, run scripts, and perform other instance customization operations during deployments.

Settings applied in the Elastic Beanstalk console override the same settings in configuration files, if they exist. This lets you have default settings in configuration files, and override them with environment-specific settings in the console. For more information about precedence, and other methods of changing settings, see Configuration options.

Configuring your .NET environment in the Elastic Beanstalk console

You can use the Elastic Beanstalk console to enable log rotation to Amazon S3, configure variables that your application can read from the environment, and change .NET framework settings.

To configure your .NET environment in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.

Container options

- **Target .NET runtime –** Set to `2.0` to run CLR v2.
- **Enable 32-bit applications –** Set to `True` to run 32-bit applications.

Log options

The Log Options section has two settings:

- **Instance profile –** Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3 –** Specifies whether log files for your application's Amazon EC2 instances are copied to the Amazon S3 bucket associated with your application.

Environment properties

The **Environment Properties** section lets you specify environment configuration settings on the Amazon EC2 instances that are running your application. These settings are passed in as key-value pairs to the application. Use `System.GetEnvironmentVariable` to read them. Identical keys can exist in both `web.config` and as environment properties. Use the `System.Configuration` namespace to read values from `web.config`.

```
NameValueCollection appConfig = ConfigurationManager.AppSettings;
string endpoint = appConfig["API_ENDPOINT"];
```

See Environment variables and other software settings for more information.

The aws:elasticbeanstalk:container:dotnet:apppool namespace

You can use a configuration file to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be platform specific or apply to all platforms in the Elastic Beanstalk service as a whole. Configuration options are organized into *namespaces*.

The .NET platform defines options in the `aws:elasticbeanstalk:container:dotnet:apppool` namespace that you can use to configure the .NET runtime.

The following example configuration file shows settings for each of the options available in this namespace:

Example .ebextensions/dotnet-settings.config

```
option_settings:
  aws:elasticbeanstalk:container:dotnet:apppool:
    Target Runtime: 2.0
    Enable 32-bit Applications: True
```

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See Configuration options for more information.

Migrating across major versions of the Elastic Beanstalk Windows server platform

AWS Elastic Beanstalk has had several major versions of its Windows Server platform. This page covers the main improvements for each major version, and what to consider before you migrate to a later version.

The Windows Server platform is currently at version 2 (v2). If your application uses any Windows Server platform version earlier than v2, we recommend that you migrate to v2.

What's new in major versions of the Windows server platform

Windows server platform V2

Version 2 (v2) of the Elastic Beanstalk Windows Server platform was released in February 2019. V2 brings the behavior of the Windows Server platform closer to that of the Elastic Beanstalk Linux-based platforms in several important ways. V2 is fully backward compatible with v1, making migration from v1 easy.

The Windows Server platform now supports the following:

- *Versioning* – Each release gets a new version number, and you can refer to past versions (that are still available to you) when creating and managing environments.
- *Enhanced health* – For details, see Enhanced health reporting and monitoring in Elastic Beanstalk.

- *Immutable* and *Rolling with an Additional Batch* deployments – For details about deployment policies, see Deploying applications to Elastic Beanstalk environments.
- *Immutable updates* – For details about update types, see Configuration changes.
- *Managed platform updates* – For details, see Managed platform updates.

Note

The new deployment and update features depend on enhanced health. Enable enhanced health to use them. For details, see Enabling Elastic Beanstalk enhanced health reporting.

Windows server platform V1

Version 1.0.0 (v1) of the Elastic Beanstalk Windows Server platform was released in October 2015. This version changes the order in which Elastic Beanstalk processes commands in configuration files during environment creation and updates.

Previous platform versions don't have a version number in the solution stack name:

- 64bit Windows Server 2012 R2 running IIS 8.5
- 64bit Windows Server Core 2012 R2 running IIS 8.5
- 64bit Windows Server 2012 running IIS 8
- 64bit Windows Server 2008 R2 running IIS 7.5

In earlier versions, the processing order for configuration files is inconsistent. During environment creation, `Container Commands` run after the application source is deployed to IIS. During a deployment to a running environment, container commands run before the new version is deployed. During a scale up, configuration files are not processed at all.

In addition to this, IIS starts up before container commands run. This behavior has led some customers to implement workarounds in container commands, pausing the IIS server before commands run, and starting it again after they complete.

Version 1 fixes the inconsistency and brings the behavior of the Windows Server platform closer to that of the Elastic Beanstalk Linux-based platforms. In the v1 platform, Elastic Beanstalk always runs container commands before starting the IIS server.

The v1 platform solution stacks have a `v1` after the Windows Server version:

- 64bit Windows Server 2012 R2 v1.1.0 running IIS 8.5
- 64bit Windows Server Core 2012 R2 v1.1.0 running IIS 8.5
- 64bit Windows Server 2012 v1.1.0 running IIS 8
- 64bit Windows Server 2008 R2 v1.1.0 running IIS 7.5

Additionally, the v1 platform extracts the contents of your application source bundle to `C:\staging\` before running container commands. After container commands complete, the contents of this folder are compressed into a .zip file and deployed to IIS. This workflow allows you to modify the contents of your application source bundle with commands or a script before deployment.

Migrating from earlier major versions of the Windows server platform

Read this section for migration considerations before you update your environment. To update your environment's platform to a newer version, see Updating your Elastic Beanstalk environment's platform version.

From V1 to V2

The Windows Server platform v2 doesn't support .NET Core 1.x and 2.0. If you're migrating your application from Windows Server v1 to v2, and your application uses one of these .NET Core versions, update your application to a .NET Core version that v2 supports. For a list of supported versions, see .NET on Windows Server with IIS in the *AWS Elastic Beanstalk Platforms*.

If your application uses a custom Amazon Machine Image (AMI), create a new custom AMI based on a Windows Server platform v2 AMI. To learn more, see Using a custom Amazon machine image (AMI) in your Elastic Beanstalk environment.

Note

The deployment and update features that are new to Windows Server v2 depend on enhanced health. When you migrate an environment to v2, enhanced health is disabled. Enable it to use these features. For details, see Enabling Elastic Beanstalk enhanced health reporting.

From pre-V1

In addition to considerations for migrating from v1, if you're migrating your application from a Windows Server solution stack that's earlier than v1, and you currently use container commands, remove any commands that you added to work around the processing inconsistencies when you migrate to a newer version. Starting with v1, container commands are guaranteed to run completely before the application source that is deployed and before IIS starts. This enables you to make any changes to the source in `C:\staging` and modify IIS configuration files during this step without issue.

For example, you can use the AWS CLI to download a DLL file to your application source from Amazon S3:

`.ebextensions\copy-dll.config`

```
container_commands:
  copy-dll:
    command: aws s3 cp s3://amzn-s3-demo-bucket/dlls/large-dll.dll .\lib\
```

For more information on using configuration files, see Advanced environment customization with configuration files (`.ebextensions`).

Running multiple applications and ASP.NET core applications with a deployment manifest

You can use a deployment manifest to tell Elastic Beanstalk how to deploy your application. By using this method, you don't need to use `MSDeploy` to generate a source bundle for a single ASP.NET application that runs at the root path of your website. Rather, you can use a manifest file to run multiple applications at different paths. Or, alternatively, you can tell Elastic Beanstalk to deploy and run the app with ASP.NET Core. You can also use a deployment manifest to configure an application pool where to run your applications.

Deployment manifests add support for .NET Core applications to Elastic Beanstalk. You can deploy a .NET Framework application without a deployment manifest. However, .NET Core applications require a deployment manifest to run on Elastic Beanstalk. When you use a deployment manifest, you create a site archive for each application, and then bundle the site archives in a second ZIP archive that contains the deployment manifest.

Deployment manifests also add the ability to run multiple applications at different paths. A deployment manifest defines an array of deployment targets, each with a site archive and a path at which IIS should run it. For example, you could run a web API at the `/api` path to serve asynchronous requests, and a web app at the root path that consumes the API.

You can use a deployment manifest to configure IIS websites with custom bindings and physical paths. This allows you to set up websites that listen on specific ports or host names before deploying your applications.

You can also use a deployment manifest to run multiple applications using application pools in IIS or Kestrel. You can configure an application pool to restart your applications periodically, run 32-bit applications, or use a specific version of the .NET Framework runtime.

For full customization, you can write your own deployment scripts in Windows PowerShell and tell Elastic Beanstalk which scripts to run to install, uninstall, and restart your application.

Deployment manifests and related features require a Windows Server platform version 1.2.0 or newer.

For detailed information about all available configuration options, properties, and advanced features like skipping IIS resets, see the deployment manifest schema reference.

Sections

.NET core apps

You can use a deployment manifest to run .NET Core applications on Elastic Beanstalk. .NET Core is a cross-platform version of .NET that comes with a command line tool (`dotnet`). You can use it to generate an application, run it locally, and prepare it for publishing.

To run a .NET Core application on Elastic Beanstalk, you can run `dotnet publish` and package the output in a ZIP archive, not including any containing directories. Place the site archive in a source bundle with a deployment manifest with a deployment target of type `aspNetCoreWeb`.

The following deployment manifest runs a .NET Core application from a site archive named `dotnet-core-app.zip` at the root path.

Example aws-windows-deployment-manifest.json - .NET core

```
{
  "manifestVersion": 1,
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "my-dotnet-core-app",
        "parameters": {
          "archive": "dotnet-core-app.zip",
          "iisPath": "/"
        }
      }
    ]
  }
}
```

Bundle the manifest and site archive in a ZIP archive to create a source bundle.

Example dotnet-core-bundle.zip

```
.
|-- aws-windows-deployment-manifest.json
`-- dotnet-core-app.zip
```

The site archive contains the compiled application code, dependencies, and `web.config` file.

Example dotnet-core-app.zip

```
.
|-- Microsoft.AspNetCore.Hosting.Abstractions.dll
|-- Microsoft.AspNetCore.Hosting.Server.Abstractions.dll
|-- Microsoft.AspNetCore.Hosting.dll
|-- Microsoft.AspNetCore.Http.Abstractions.dll
|-- Microsoft.AspNetCore.Http.Extensions.dll
|-- Microsoft.AspNetCore.Http.Features.dll
|-- Microsoft.AspNetCore.Http.dll
|-- Microsoft.AspNetCore.HttpOverrides.dll
|-- Microsoft.AspNetCore.Server.IISIntegration.dll
|-- Microsoft.AspNetCore.Server.Kestrel.dll
|-- Microsoft.AspNetCore.WebUtilities.dll
|-- Microsoft.Extensions.Configuration.Abstractions.dll
|-- Microsoft.Extensions.Configuration.EnvironmentVariables.dll
|-- Microsoft.Extensions.Configuration.dll
|-- Microsoft.Extensions.DependencyInjection.Abstractions.dll
|-- Microsoft.Extensions.DependencyInjection.dll
|-- Microsoft.Extensions.FileProviders.Abstractions.dll
|-- Microsoft.Extensions.FileProviders.Physical.dll
|-- Microsoft.Extensions.FileSystemGlobbing.dll
|-- Microsoft.Extensions.Logging.Abstractions.dll
|-- Microsoft.Extensions.Logging.dll
|-- Microsoft.Extensions.ObjectPool.dll
|-- Microsoft.Extensions.Options.dll
|-- Microsoft.Extensions.PlatformAbstractions.dll
|-- Microsoft.Extensions.Primitives.dll
|-- Microsoft.Net.Http.Headers.dll
|-- System.Diagnostics.Contracts.dll
|-- System.Net.WebSockets.dll
|-- System.Text.Encodings.Web.dll
|-- dotnet-core-app.deps.json
|-- dotnet-core-app.dll
|-- dotnet-core-app.pdb
|-- dotnet-core-app.runtimeconfig.json
`-- web.config
```

Run multiple applications

You can run multiple applications with a deployment manifest by defining multiple deployment targets.

The following deployment manifest configures two .NET Core applications. The `WebApiSampleApp` application implements a simple web API and serves asynchronous requests at the `/api` path. The `DotNetSampleApp` application is a web application that serves requests at the root path.

Example aws-windows-deployment-manifest.json - multiple apps

```
{
  "manifestVersion": 1,
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "WebAPISample",
        "parameters": {
          "appBundle": "WebApiSampleApp.zip",
          "iisPath": "/api"
        }
      },
      {
        "name": "DotNetSample",
        "parameters": {
          "appBundle": "DotNetSampleApp.zip",
          "iisPath": "/"
        }
      }
    ]
  }
}
```

A sample application with multiple applications is available here:

- **Deployable source bundle** - dotnet-multiapp-sample-bundle-v2.zip
- **Source code** - dotnet-multiapp-sample-source-v2.zip

Configure IIS websites

You can configure IIS websites with custom bindings and physical paths using the deployment manifest. This is useful when you need to set up websites that listen on specific ports, use custom host names, or serve content from specific directories.

The following deployment manifest configures a custom IIS website that listens on HTTP with a specific port number and a custom physical path:

Example aws-windows-deployment-manifest.json - IIS website configuration

```
{
  "manifestVersion": 1,
  "iisConfig": {
    "websites": [
      {
        "name": "MyCustomSite",
        "physicalPath": "C:\inetpub\wwwroot\mysite",
        "bindings": [
          {
            "protocol": "http",
            "port": 8080,
            "hostName": "mysite.local"
          }
        ]
      }
    ]
  },
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "my-dotnet-core-app",
        "parameters": {
          "appBundle": "dotnet-core-app.zip",
          "iisWebSite": "MyCustomSite",
          "iisPath": "/"
        }
      }
    ]
  }
}
```

In this example:

- A website named "MyCustomSite" is created with a custom physical path
- The website has an HTTP binding on port 8080 with a specific host name
- The ASP.NET Core application is deployed to this custom website using the `iisWebSite` parameter

Using Application Request Routing (ARR)

Application Request Routing (ARR) and URL Rewrite modules are pre-installed and available in Elastic Beanstalk Windows AMIs. These modules enable advanced routing scenarios and URL manipulation through IIS configuration using ebextensions or application configuration.

The following example shows a simple deployment manifest that configures a website with a custom port, combined with an ebextensions configuration that sets up basic ARR routing:

Example aws-windows-deployment-manifest.json - Simple ARR setup

```
{
  "manifestVersion": 1,
  "iisConfig": {
    "websites": [
      {
        "name": "ARRSite",
        "physicalPath": "C:\\inetpub\\wwwroot\\arrsite",
        "bindings": [
          {
            "protocol": "http",
            "port": 8080,
            "hostName": "localhost"
          }
        ]
      }
    ]
  },
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "BackendApp",
        "parameters": {
          "appBundle": "backend-app.zip",
          "iisWebSite": "ARRSite",
          "iisPath": "/backend"
        }
      }
    ]
  }
}
```

ARR configuration is done through ebextensions. The following configuration sets up basic ARR routing rules:

Example .ebextensions/arr-config.config - Basic ARR configuration

```
files:
  "C:\\temp\\configure-arr.ps1":
    content: |
      # Enable ARR proxy at server level
      Set-WebConfigurationProperty -PSPath 'MACHINE/WEBROOT/APPHOST' -Filter
 'system.webServer/proxy' -Name 'enabled' -Value 'True'

      # Clear any existing global rules to avoid conflicts
      Clear-WebConfiguration -PSPath 'MACHINE/WEBROOT/APPHOST' -Filter
 'system.webServer/rewrite/globalRules'

      # Add global rule to route all requests to backend
      Add-WebConfigurationProperty -PSPath 'MACHINE/WEBROOT/APPHOST' `
        -Filter 'system.webServer/rewrite/globalRules' `
        -Name '.' `
        -Value @{
          name = 'Route_to_Backend'
          stopProcessing = 'True'
          match = @{ url = '^(?!backend/)(.*)' }
          action = @{
            type = 'Rewrite'
            url = 'http://localhost:8080/backend/{R:1}'
          }
        }

container_commands:
  01_configure_arr:
    command: powershell -ExecutionPolicy Bypass -File "C:\\temp\\configure-arr.ps1"
    waitAfterCompletion: 0
```

This configuration creates a website on port 8080 and sets up ARR to route all incoming requests to the backend application running on that site.

Configure application pools

You can support multiple applications in your Windows environment. Two approaches are available:

- You can use the out-of-process hosting model with the Kestrel web server. With this model, you configure multiple applications to run in one application pool.
- You can use the in-process hosting model.With this model, you use multiple application pools to run multiple applications with only one application in each pool. If you're using IIS server and need to run multiple applications, you must use this approach.

To configure Kestrel to run multiple applications in one application pool, add `hostingModel="OutofProcess"` in the `web.config` file. Consider the following examples.

Example web.config - for Kestrel out-of-process hosting model

```
<configuration>
<location path="." inheritInChildApplications="false">
<system.webServer>
<handlers>
<add
    name="aspNetCore"
    path="*" verb="*"
    modules="AspNetCoreModuleV2"
    resourceType="Unspecified" />
</handlers>
<aspNetCore
    processPath="dotnet"
    arguments=".\CoreWebApp-5-0.dll"
    stdoutLogEnabled="false"
    stdoutLogFile=".\logs\stdout"
    hostingModel="OutofProcess" />
</system.webServer>
</location>
</configuration>
```

Example aws-windows-deployment-manifest.json - multiple applications

```
{
"manifestVersion": 1,
  "deployments": {"msDeploy": [
      {"name": "Web-app1",
        "parameters": {"archive": "site1.zip",
          "iisPath": "/"
        }
      },
      {"name": "Web-app2",
        "parameters": {"archive": "site2.zip",
          "iisPath": "/app2"
        }
      }
    ]
  }
}
```

IIS doesn't support multiple applications in one application pool because it uses the in-process hosting model. Therefore, you need to configure multiple applications by assigning each application to one application pool. In other words, assign only one application to one application pool.

You can configure IIS to use different application pools in the `aws-windows-deployment-manifest.json` file. Make the following updates as you refer to the next example file:

- Add an `iisConfig` section that includes a subsection called `appPools`.
- In the `appPools` block, list the application pools.
- In the `deployments` section, define a `parameters` section for each application.
- For each application the `parameters` section specifies an archive, a path to run it, and an `appPool` in which to run.

The following deployment manifest configures two application pools that restart their application every 10 minutes. They also attach their applications to a .NET Framework web application that runs at the path specified.

Example aws-windows-deployment-manifest.json - one application per application pool

```
{
"manifestVersion": 1,
  "iisConfig": {"appPools": [
      {"name": "MyFirstPool",
       "recycling": {"regularTimeInterval": 10}
      },
      {"name": "MySecondPool",
       "recycling": {"regularTimeInterval": 10}
      }
     ]
    },
  "deployments": {"msDeploy": [
      {"name": "Web-app1",
        "parameters": {
           "archive": "site1.zip",
           "iisPath": "/",
           "appPool": "MyFirstPool"
           }
      },
      {"name": "Web-app2",
        "parameters": {
           "archive": "site2.zip",
           "iisPath": "/app2",
           "appPool": "MySecondPool"
          }
      }
     ]
    }
}
```

Define custom deployments

For even more control, you can completely customize an application deployment by defining a *custom deployment*.

This deployment manifest instructs Elastic Beanstalk to execute PowerShell scripts in 32-bit mode. It specifies three scripts: an `install` script (`siteInstall.ps1`) that runs during instance launch and deployments, an `uninstall` script (`siteUninstall.ps1`) that executes before installing new versions during deployments, and a `restart` script (`siteRestart.ps1`) that runs when you select Restart App Server in the AWS management console.

Example aws-windows-deployment-manifest.json - custom deployment

```
{
  "manifestVersion": 1,
  "deployments": {
    "custom": [
      {
        "name": "Custom site",
        "architecture" : 32,
        "scripts": {
          "install": {
            "file": "siteInstall.ps1"
          },
          "restart": {
            "file": "siteRestart.ps1"
          },
          "uninstall": {
            "file": "siteUninstall.ps1"
          }
        }
      }
    ]
  }
}
```

Include any artifacts required to run the application in your source bundle with the manifest and scripts.

Example Custom-site-bundle.zip

```
.
|-- aws-windows-deployment-manifest.json
|-- siteInstall.ps1
|-- siteRestart.ps1
|-- siteUninstall.ps1
`-- site-contents.zip
```

Deployment manifest schema reference

The deployment manifest is a JSON file that defines how Elastic Beanstalk should deploy and configure your Windows applications. This section provides a comprehensive reference for all supported properties and configuration options in the manifest schema.

Manifest structure

The deployment manifest follows a specific JSON schema with the following top-level structure:

Example Basic manifest structure

```
{
  "manifestVersion": 1,
  "skipIISReset": false,
  "iisConfig": {
    "websites": [...],
    "appPools": [...]
  },
  "deployments": {
    "msDeploy": [...],
    "aspNetCoreWeb": [...],
    "custom": [...]
  }
}
```

Top-level properties

`manifestVersion` (required)

Type: Number

Default: 1

Valid values: 1

Specifies the version of the manifest schema. Currently, only version 1 is supported.

`skipIISReset` (optional)

Type: Boolean

Default: false

Controls whether IIS is reset during application deployments. This flag affects both `msDeploy` and `aspNetCoreWeb` deployment types.

Behavior:

- *Not specified or `false` (default):* IIS resets are performed during install, uninstall, and update operations. This is the traditional behavior.
- *`true`:* IIS resets are skipped during deployment operations.

Benefits:

- *Reduced downtime* – Applications experience shorter service interruptions during deployments.
- *Faster deployments* – Eliminates the time required for IIS to fully restart and reinitialize.

Note

When using `skipIISReset`, the RestartAppServer operation performs an IIS reset regardless of this flag setting.

Example:

```
{
  "manifestVersion": 1,
  "skipIISReset": true,
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "my-dotnet-core-app",
        "parameters": {
          "archive": "dotnet-core-app.zip",
          "iisPath": "/"
        }
      }
    ]
  }
}
```

`deployments` (required)

Type: Object

Contains the deployment configurations for your applications. This object can include `msDeploy`, `aspNetCoreWeb`, and `custom` deployment types.

`iisConfig` (optional)

Type: Object

Defines IIS configuration settings to apply before deploying applications. Supports both website and application pool configuration.

IIS configuration

The `iisConfig` section allows you to configure IIS settings before deploying your applications. This includes setting up application pools with specific configurations and configuring IIS websites with custom bindings.

IIS websites

IIS websites allow you to configure custom website settings including physical paths and network bindings before deploying your applications.

Important considerations for creating different IIS websites

- *Website setup order:* Websites are configured sequentially in the order they appear in the `websites` array. The platform processes each website configuration in sequence, so ensure proper ordering if you have dependencies between websites.
- *Firewall and port access:* Only port 80 is automatically exposed through the default Elastic Beanstalk Windows firewall configuration. If you configure websites to use non-

standard ports, you must define custom firewall rules through ebextensions or custom deployment scripts to allow external access to these ports.

Example Website configuration

```
{
  "iisConfig": {
    "websites": [
      {
        "name": "MyCustomSite",
        "physicalPath": "C:\inetpub\wwwroot\mysite",
        "bindings": [
          {
            "protocol": "http",
            "port": 8080,
            "hostName": "mysite.local"
          },
          {
            "protocol": "https",
            "port": 8443
          }
        ]
      }
    ]
  }
}
```

Website properties

`name` (required)

Type: String

The name of the IIS website. This name is used to identify the website in IIS Manager and must be unique within the IIS configuration.

`physicalPath` (required)

Type: String

The physical path on the server where the website files are stored. This path must be accessible to the IIS worker process.

`bindings` (required)

Type: Array

Minimum items: 1

An array of binding configurations that define how the website responds to network requests. Each binding specifies a protocol, port, and optional host name.

Website bindings

Website bindings define the network endpoints where your IIS website will listen for incoming requests.

`protocol` (required)

Type: String

Valid values: "http", "https"

The protocol used for the binding.

`port` (required)

Type: Integer

Valid range: 1-65535

The port number on which the website will listen for requests.

`hostName` (optional)

Type: String

The host name (domain name) for the binding.

Application pools

Application pools provide isolation between applications and allow you to configure runtime settings for groups of applications.

Example Application pool configuration

```
{
  "iisConfig": {
    "appPools": [
      {
        "name": "MyAppPool",
        "enable32Bit": false,
        "managedPipelineMode": "Integrated",
        "managedRuntimeVersion": "v4.0",
        "queueLength": 1000,
        "cpu": {
          "limitPercentage": 80,
          "limitAction": "Throttle",
          "limitMonitoringInterval": 5
        },
        "recycling": {
          "regularTimeInterval": 1440,
          "requestLimit": 10000,
          "memory": 1048576,
          "privateMemory": 524288
        }
      }
    ]
  }
}
```

Application pool properties

`name` (required)

Type: String

The name of the application pool. This name is used to reference the pool in deployment configurations.

`enable32Bit` (optional)

Type: Boolean

Enables a 32-bit application to run on a 64-bit version of Windows. Set to `true` for legacy applications that require 32-bit compatibility.

`managedPipelineMode` (optional)

Type: String

Valid values: "Integrated", "Classic"

Specifies the request-processing mode for the application pool.

`managedRuntimeVersion` (optional)

Type: String

Valid values: "No Managed Code", "v2.0", "v4.0"

Specifies the .NET Framework version for the application pool.

`queueLength` (optional)

Type: Integer

Maximum number of requests that HTTP.sys queues for the application pool before rejecting additional requests.

CPU configuration

The `cpu` object configures CPU usage limits and monitoring for the application pool.

`limitPercentage` (optional)

Type: Number

Maximum percentage of CPU time that worker processes in the application pool can consume.

`limitAction` (optional)

Type: String

Valid values: "NoAction", "KillW3wp", "Throttle", "ThrottleUnderLoad"

Action to take when the CPU limit is reached.

`limitMonitoringInterval` (optional)

Type: Number

Reset period (in minutes) for CPU monitoring and throttling limits.

Recycling configuration

The `recycling` object configures when and how application pool worker processes are recycled.

`regularTimeInterval` (optional)

Type: Integer

Time interval (in minutes) after which the application pool recycles. Set to 0 to disable time-based recycling.

`requestLimit` (optional)

Type: Integer

Maximum number of requests the application pool processes before recycling.

`memory` (optional)

Type: Integer

Amount of virtual memory (in kilobytes) that triggers worker process recycling.

`privateMemory` (optional)

Type: Integer

Amount of private memory (in kilobytes) that triggers worker process recycling.

Deployment types

The `deployments` object contains arrays of deployment configurations for different application types. Each deployment type has specific properties and use cases.

MSDeploy deployments

MSDeploy deployments are used for traditional .NET Framework applications that can be deployed using Web Deploy (MSDeploy).

Example MSDeploy deployment configuration

```
{
  "deployments": {
    "msDeploy": [
      {
        "name": "WebApp",
        "description": "Main web application",
        "parameters": {
          "appBundle": "webapp.zip",
          "iisPath": "/",
          "appPool": "DefaultAppPool"
        }
      }
    ]
  }
}
```

MSDeploy deployment properties

`name` (required)

Type: String

Unique name for the deployment. This name must be unique across all deployments in the manifest.

`description` (optional)

Type: String

Human-readable description of the deployment.

`parameters` (required)

Type: Object

Configuration parameters for the MSDeploy operation.

`scripts` (optional)

Type: Object

PowerShell scripts to run at various stages of the deployment lifecycle.

MSDeploy parameters

`appBundle` (required)

Type: String

Path to the application bundle (ZIP file) relative to the manifest file. This bundle contains the application files to deploy.

`iisWebSite` (optional)

Type: String

Default: "Default Web Site"

The IIS website to deploy the application to. By default, applications are deployed to the "Default Web Site". Optionally, you can specify a different website name, such as one configured in the `iisConfig.websites` section.

`iisPath` (optional)

Type: String

Default: "/"

Virtual directory path in IIS where the application will be deployed. Use "/" for the root path or "/api" for a subdirectory.

`appPool` (optional)

Type: String

Name of the application pool to run this application.

ASP.NET Core deployments

ASP.NET Core deployments are specifically designed for .NET Core and .NET 5+ applications.

Example ASP.NET Core deployment configuration

```
{
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "CoreAPI",
        "description": "ASP.NET Core Web API",
        "parameters": {
          "appBundle": "coreapi.zip",
          "iisPath": "/api",
          "appPool": "CoreAppPool"
        }
      }
    ]
  }
}
```

ASP.NET Core deployments use the same property structure as MSDeploy deployments, with the key difference being the runtime environment and hosting model used for the application.

ASP.NET Core deployment parameters

`appBundle` (required)

Type: String

Path to the application bundle relative to the manifest file. This can be either a ZIP archive or a directory path containing the published ASP.NET Core application.

`iisWebSite` (optional)

Type: String

Default: "Default Web Site"

The IIS website to deploy the ASP.NET Core application to. By default, applications are deployed to the "Default Web Site". Optionally, you can specify a different website name, such as one configured in the `iisConfig.websites` section.

`iisPath` (optional)

Type: String

Default: "/"

Virtual directory path in IIS for the ASP.NET Core application.

`appPool` (optional)

Type: String

Application pool for the ASP.NET Core application. The pool will be configured appropriately for ASP.NET Core hosting.

Custom deployments

Custom deployments provide complete control over the deployment process through PowerShell scripts. This deployment type is useful for complex scenarios that require custom installation, configuration, or deployment logic.

Example Custom deployment configuration

```
{
  "deployments": {
    "custom": [
      {
        "name": "CustomService",
        "description": "Custom Windows service deployment",
        "architecture": 32,
        "scripts": {
          "install": {
            "file": "install-service.ps1"
          },
          "restart": {
            "file": "restart-service.ps1"
          },
          "uninstall": {
            "file": "uninstall-service.ps1",
            "ignoreErrors": true
          }
        }
      }
    ]
  }
}
```

Custom deployment properties

`name` (required)

Type: String

Unique name for the custom deployment.

`description` (optional)

Type: String

Description of the custom deployment.

`architecture` (optional)

Type: Integer

Default: 32

Valid values: 32, 64

The architecture specification for execution mode of powershell scripts

`scripts` (required)

Type: Object

PowerShell scripts that define the deployment behavior. Custom deployments support additional script types compared to other deployment types.

Deployment scripts

Deployment scripts are PowerShell scripts that run at specific points during the deployment lifecycle. Different deployment types support different sets of script events.

Script events

The following script events are available depending on the deployment type:

Standard deployment scripts (msDeploy and aspNetCoreWeb)

`preInstall`

Runs before the application is installed or updated.

`postInstall`

Runs after the application is installed or updated.

`preRestart`

Runs before the application is restarted.

`postRestart`

Runs after the application is restarted.

`preUninstall`

Runs before the application is uninstalled.

`postUninstall`

Runs after the application is uninstalled.

Custom deployment scripts (custom deployments only)

`install`

Primary installation script for custom deployments. This script is responsible for installing the application or service.

`restart`

Script to restart the application or service. Called when the environment is restarted.

`uninstall`

Script to uninstall the application or service. Called during environment termination or application removal.

Script properties

Each script is defined as an object with the following properties:

`file` (required)

Type: String

Path to the PowerShell script file relative to the manifest file. The script should have a `.ps1` extension.

`ignoreErrors` (optional)

Type: Boolean

Default: false

When set to `true`, deployment continues even if the script fails. Use this for non-critical scripts or cleanup operations.

Example Script configuration example

```
{
  "scripts": {
    "preInstall": {
      "file": "backup-config.ps1",
      "ignoreErrors": true
    },
    "postInstall": {
      "file": "configure-app.ps1"
    }
  }
}
```

Using EC2 Fast Launch with Windows platform branches

The EC2 Fast Launch feature reduces Windows instance launch times in your Elastic Beanstalk environments. The purpose of this topic is to guide you on using this feature with your Elastic Beanstalk environments. Starting with Windows platform version 2.16.2, released on January 22, 2025, Elastic Beanstalk platform releases include base AMIs with EC2 Fast Launch enabled.

Default EC2 Fast Launch availability

The latest Elastic Beanstalk Windows platform versions include base AMIs with EC2 Fast Launch automatically enabled, with no additional costs. However, when newer platform versions are released, EC2 Fast Launch may not remain automatically enabled on base AMIs from older platform versions.

We recommend upgrading to the latest Windows platform version to use base AMIs with EC2 Fast Launch automatically enabled. However, if you need to continue using your existing platform version, you can manually enable EC2 Fast Launch on your environment's base AMI. For instructions, see Manually configuring EC2 Fast Launch.

Manually configuring EC2 Fast Launch

Note

Manually enabling EC2 Fast Launch may incur additional costs compared to using platform versions with EC2 Fast Launch automatically enabled. For more information about EC2 Fast Launch costs, see the Manage costs for EC2 Fast Launch underlying resources page in the *Amazon EC2 User Guide*.

Follow these steps to enable EC2 Fast Launch on a Windows base AMI used by your Elastic Beanstalk environment:

To manually enable EC2 Fast Launch for your Elastic Beanstalk environment

1. Identify your environment's base AMI:

   Follow the steps in Creating a Custom AMI to identify your environment's base AMI ID. Note that you don't need to create a custom AMI - you only need to follow the steps to locate your current base AMI ID.
2. Enable EC2 Fast Launch on the AMI:

   Use the instructions in Enable EC2 Fast Launch in the *Amazon EC2 User Guide* to configure EC2 Fast Launch for your AMI.

Adding an Amazon RDS DB instance to your .NET application environment

This topic provides instructions to create an Amazon RDS using the Elastic Beanstalk console. You can use an Amazon Relational Database Service (Amazon RDS) DB instance to store data gathered and modified by your application. The database can be coupled to your environment and managed by Elastic Beanstalk, or it can be created as decoupled and managed externally by another service. In these instructions the database is coupled to your environment and managed by Elastic Beanstalk. For more information about integrating an Amazon RDS with Elastic Beanstalk, see Adding a database to your Elastic Beanstalk environment.

Sections

Adding a DB instance to your environment

To add a DB instance to your environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.

4. In the **Database** configuration category, choose **Edit**.
5. Choose a DB engine, and enter a user name and password.
6. To save the changes choose **Apply** at the bottom of the page.

Adding a DB instance takes about 10 minutes. When the environment update is complete, the DB instance's hostname and other connection information are available to your application through the following environment properties:

Property name	Description	Property value
`RDS_HOSTNAME`	The hostname of the DB instance.	On the **Connectivity & security** tab on the Amazon RDS console: **Endpoint**.
`RDS_PORT`	The port where the DB instance accepts connectio ns. The default value varies among DB engines.	On the **Connectivity & security** tab on the Amazon RDS console: **Port**.
`RDS_DB_NAME`	The database name, **ebdb**.	On the **Configuration** tab on the Amazon RDS console: **DB Name**.
`RDS_USERNAME`	The username that you configured for your database.	On the **Configuration** tab on the Amazon RDS console: **Master username**.
`RDS_PASSWORD`	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

For more information about configuring a database instance coupled with an Elastic Beanstalk environment, see Adding a database to your Elastic Beanstalk environment.

Downloading a driver

Download and install the `EntityFramework` package and a database driver for your development environment with `NuGet`.

Common entity framework database providers for .NET

- **SQL Server –** `Microsoft.EntityFrameworkCore.SqlServer`
- **MySQL –** `Pomelo.EntityFrameworkCore.MySql`
- **PostgreSQL –** `Npgsql.EntityFrameworkCore.PostgreSQL`

Connecting to a database

Elastic Beanstalk provides connection information for attached DB instances in environment properties. Use `ConfigurationManager.AppSettings` to read the properties and configure a database connection.

Example Helpers.cs - connection string method

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Linq;
using System.Web;

namespace MVC5App.Models
{
  public class Helpers
  {
    public static string GetRDSConnectionString()
    {
      var appConfig = ConfigurationManager.AppSettings;

      string dbname = appConfig["RDS_DB_NAME"];

      if (string.IsNullOrEmpty(dbname)) return null;

      string username = appConfig["RDS_USERNAME"];
      string password = appConfig["RDS_PASSWORD"];
      string hostname = appConfig["RDS_HOSTNAME"];
      string port = appConfig["RDS_PORT"];

      return "Data Source=" + hostname + ";Initial Catalog=" + dbname + ";User ID=" +
 username + ";Password=" + password + ";";
    }
  }
}
```

Use the connection string to initialize your database context.

Example DBContext.cs

```
using System.Data.Entity;
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;

namespace MVC5App.Models
{
  public class RDSContext : DbContext
  {
    public RDSContext()
      : base(GetRDSConnectionString())
    {
    }

    public static RDSContext Create()
    {
      return new RDSContext();
    }
  }
}
```

The AWS Toolkit for Visual Studio

Visual Studio provides templates for different programming languages and application types. You can start with any of these templates. The AWS Toolkit for Visual Studio also provides three project templates that bootstrap development of your application: AWS Console Project, AWS Web Project, and AWS Empty Project. For this example, you'll create a new ASP.NET Web Application.

To create a new ASP.NET web application project

1. In Visual Studio, on the **File** menu, click **New** and then click **Project**.
2. In the **New Project** dialog box, click **Installed Templates**, click **Visual C#**, and then click **Web**. Click **ASP.NET Empty Web Application**, type a project name, and then click **OK**.

To run a project

Do one of the following:

1. Press **F5**.
2. Select **Start Debugging** from the **Debug** menu.

Test locally

Visual Studio makes it easy for you to test your application locally. To test or run ASP.NET web applications, you need a web server. Visual Studio offers several options, such as Internet Information Services (IIS), IIS Express, or the built-in Visual Studio Development Server. To learn about each of these options and to decide which one is best for you, see Web Servers in Visual Studio for ASP.NET Web Projects.

Create an Elastic Beanstalk environment

After testing your application, you are ready to deploy it to Elastic Beanstalk.

Note

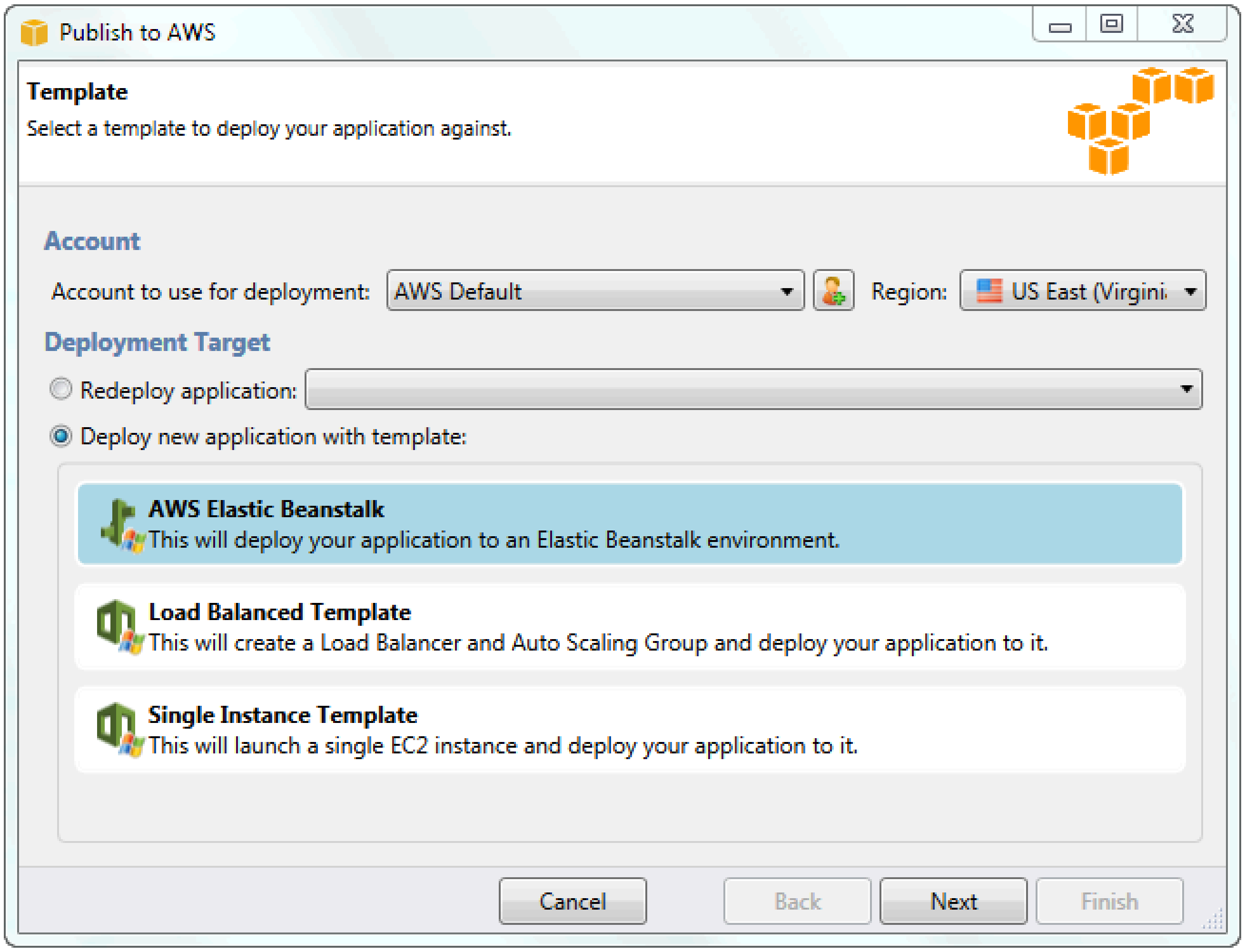
Configuration file needs to be part of the project to be included in the archive. Alternatively, instead of including the configuration files in the project, you can use Visual Studio to deploy all files in the project folder. In **Solution Explorer**, right-click the project name, and then click **Properties**. Click the **Package/Publish Web** tab. In the **Items to deploy** section, select **All Files in the Project Folder** in the drop-down list.

To deploy your application to Elastic Beanstalk using the AWS toolkit for Visual Studio

1. In **Solution Explorer**, right-click your application and then select **Publish to AWS**.
2. In the **Publish to AWS** wizard, enter your account information.
   a. For **AWS account to use for deployment**, select your account or select **Other** to enter new account information.
   b. For **Region**, select the region where you want to deploy your application. For information about available AWS Regions, see AWS Elastic Beanstalk Endpoints and Quotas in the *AWS*

General Reference. If you select a region that is not supported by Elastic Beanstalk, then the option to deploy to Elastic Beanstalk will become unavailable.

c. Click **Deploy new application with template** and select **Elastic Beanstalk**. Then click **Next**.



Publish to AWS

Template

Select a template to deploy your application against.

Account

Account to use for deployment: AWS Default

Region: US East (Virgini

Deployment Target

( ) Redeploy application:

(●) Deploy new application with template:

AWS Elastic Beanstalk
This will deploy your application to an Elastic Beanstalk environment.

Load Balanced Template
This will create a Load Balancer and Auto Scaling Group and deploy your application to it.

Single Instance Template
This will launch a single EC2 instance and deploy your application to it.

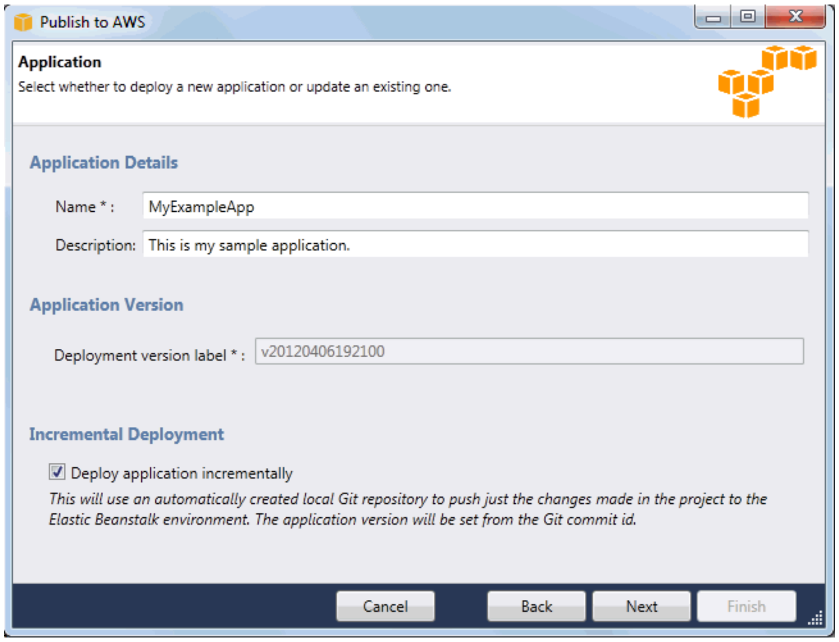
3. On the **Application** page, enter your application details.

a. For **Name**, type the name of the application.

b. For **Description**, type a description of the application. This step is optional.

c. The version label of the application automatically appears in the **Deployment version label**.

d. Select **Deploy application incrementally** to deploy only the changed files. An incremental deployment is faster because you are updating only the files that changed instead of all the files. If you choose this option, an application version will be set from the Git commit ID. If you choose to not deploy your application incrementally, then you can update the version label in the **Deployment version label** box.


Publish to AWS

Application

Select whether to deploy a new application or update an existing one.

Application Details

Name * : MyExampleApp

Description: This is my sample application.

Application Version

Deployment version label * : v20120406192100

Incremental Deployment

☑ Deploy application incrementally

This will use an automatically created local Git repository to push just the changes made in the project to the Elastic Beanstalk environment. The application version will be set from the Git commit id.

   e. Click **Next**.

4. On the **Environment** page, describe your environment details.

   a. Select **Create a new environment for this application**.

   b. For **Name**, type a name for your environment.

   c. For **Description**, characterize your environment. This step is optional.

   d. Select the **Type** of environment that you want.

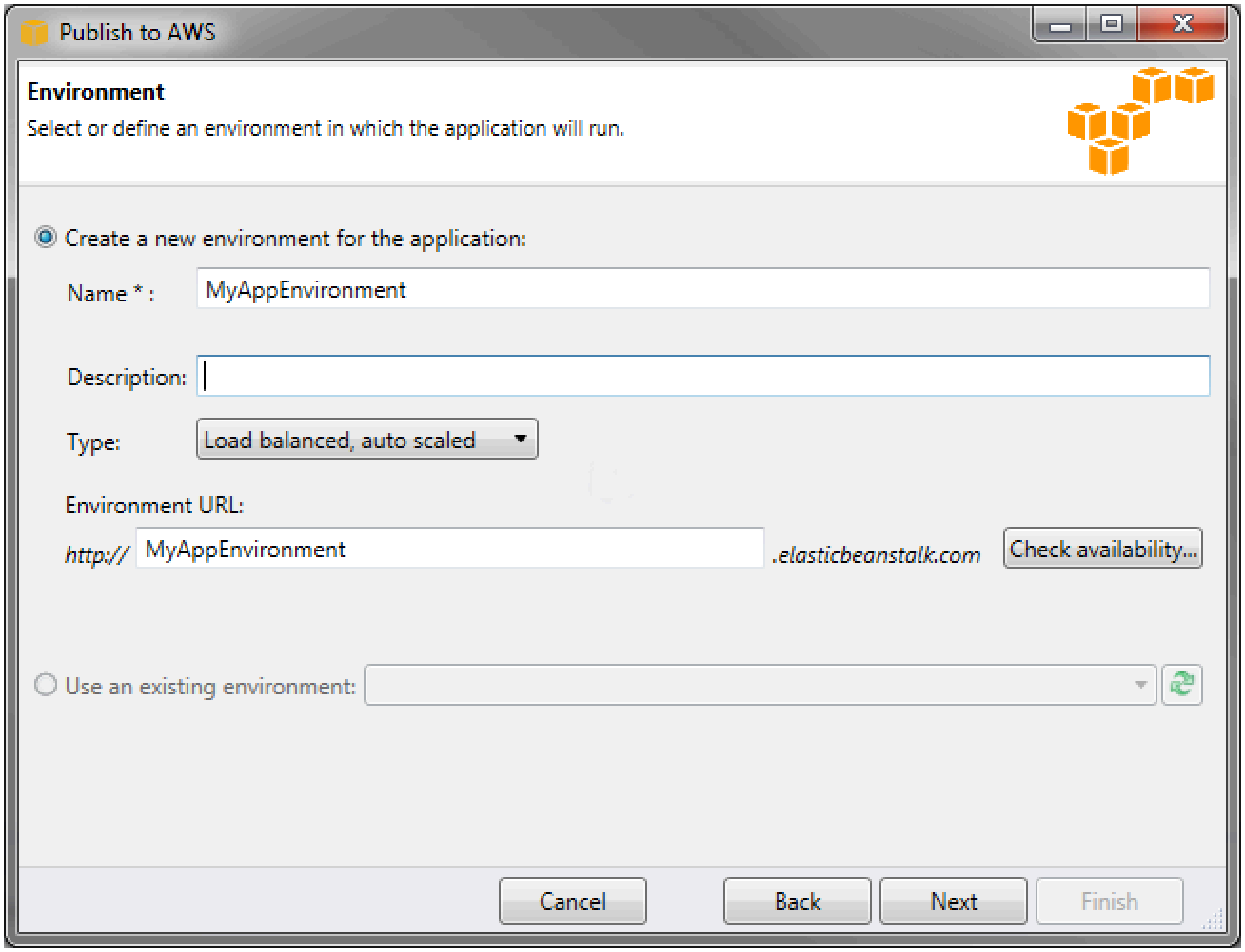
   You can select either **Load balanced, auto scaled** or a **Single instance** environment. For more information, see Environment types.

   Note

   For single-instance environments, load balancing, auto scaling, and the health check URL settings don't apply.

   e. The environment URL automatically appears in the **Environment URL** once you move your cursor to that box.

   f. Click **Check availability** to make sure the environment URL is available.



Publish to AWS

Environment

Select or define an environment in which the application will run.

(●) Create a new environment for the application:

Name * : MyAppEnvironment

Description:

Type: Load balanced, auto scaled

Environment URL:

http:// MyAppEnvironment .elasticbeanstalk.com

( ) Use an existing environment:

g. Click **Next**.

5. On the **AWS Options** page, configure additional options and security information for your deployment.

   a. For **Container Type**, select **64bit Windows Server 2012 running IIS 8** or **64bit Windows Server 2008 running IIS 7.5**.

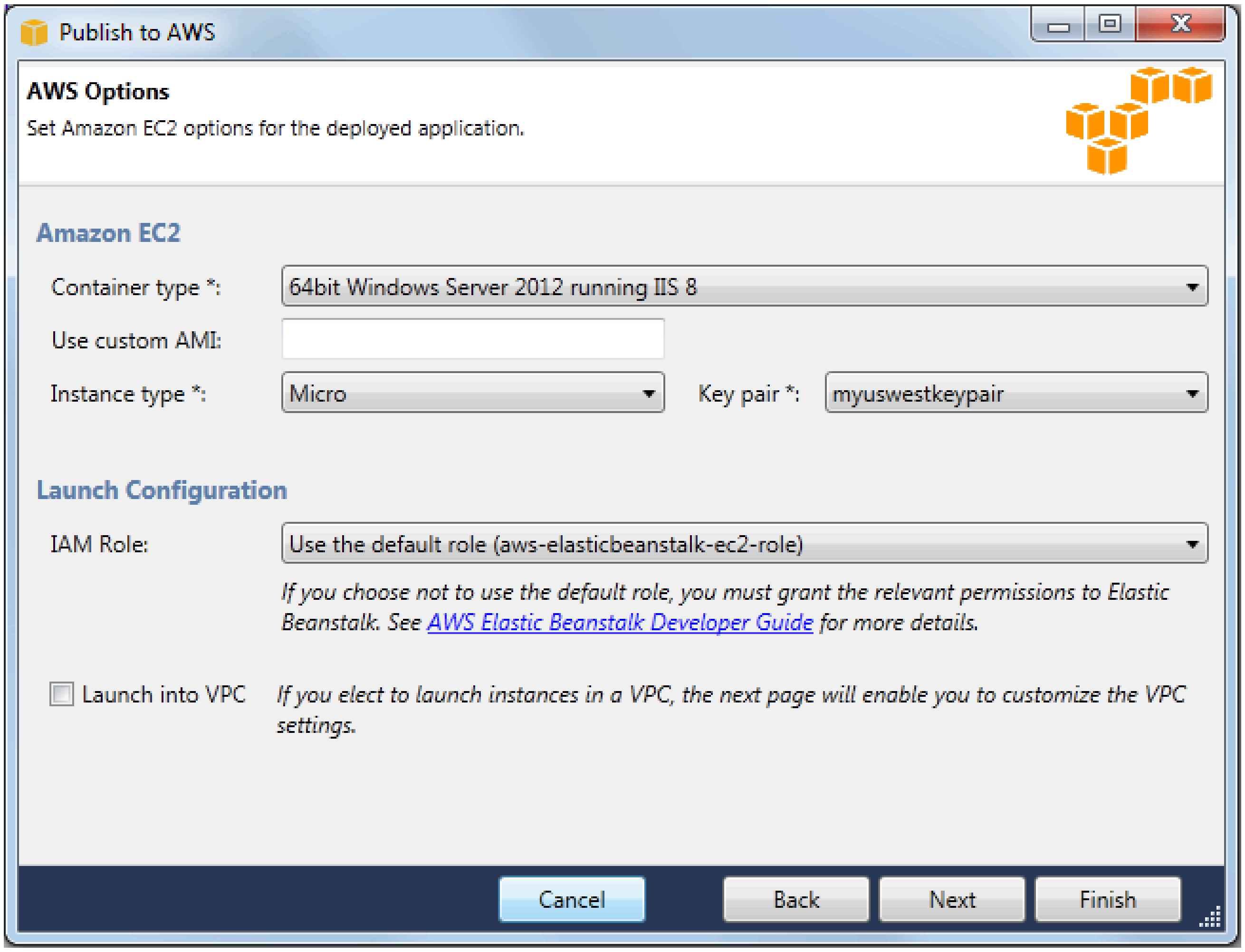
   b. For **Instance Type**, select **Micro**.

   c. For **Key pair**, select **Create new key pair**. Type a name for the new key pair—in this example, we use **`myuswestkeypair`**—and then click **OK**. A key pair enables remote-desktop access to your Amazon EC2 instances. For more information on Amazon EC2 key pairs, see Using Credentials in the *Amazon Elastic Compute Cloud User Guide*.

   d. Select an instance profile.

If you do not have an instance profile, select **Create a default instance profile**. For information about using instance profiles with Elastic Beanstalk, see Managing Elastic Beanstalk instance profiles.

e. If you have a custom VPC that you would like to use with your environment, click **Launch into VPC**. You can configure the VPC information on the next page. For more information about Amazon VPC, go to Amazon Virtual Private Cloud (Amazon VPC). For a list of supported nonlegacy container types, see the section called “Why are some platform versions marked legacy?”



Publish to AWS

AWS Options

Set Amazon EC2 options for the deployed application.

Amazon EC2

Container type *: 64bit Windows Server 2012 running IIS 8

Use custom AMI:

Instance type *: Micro

Key pair *: myuswestkeypair

Launch Configuration

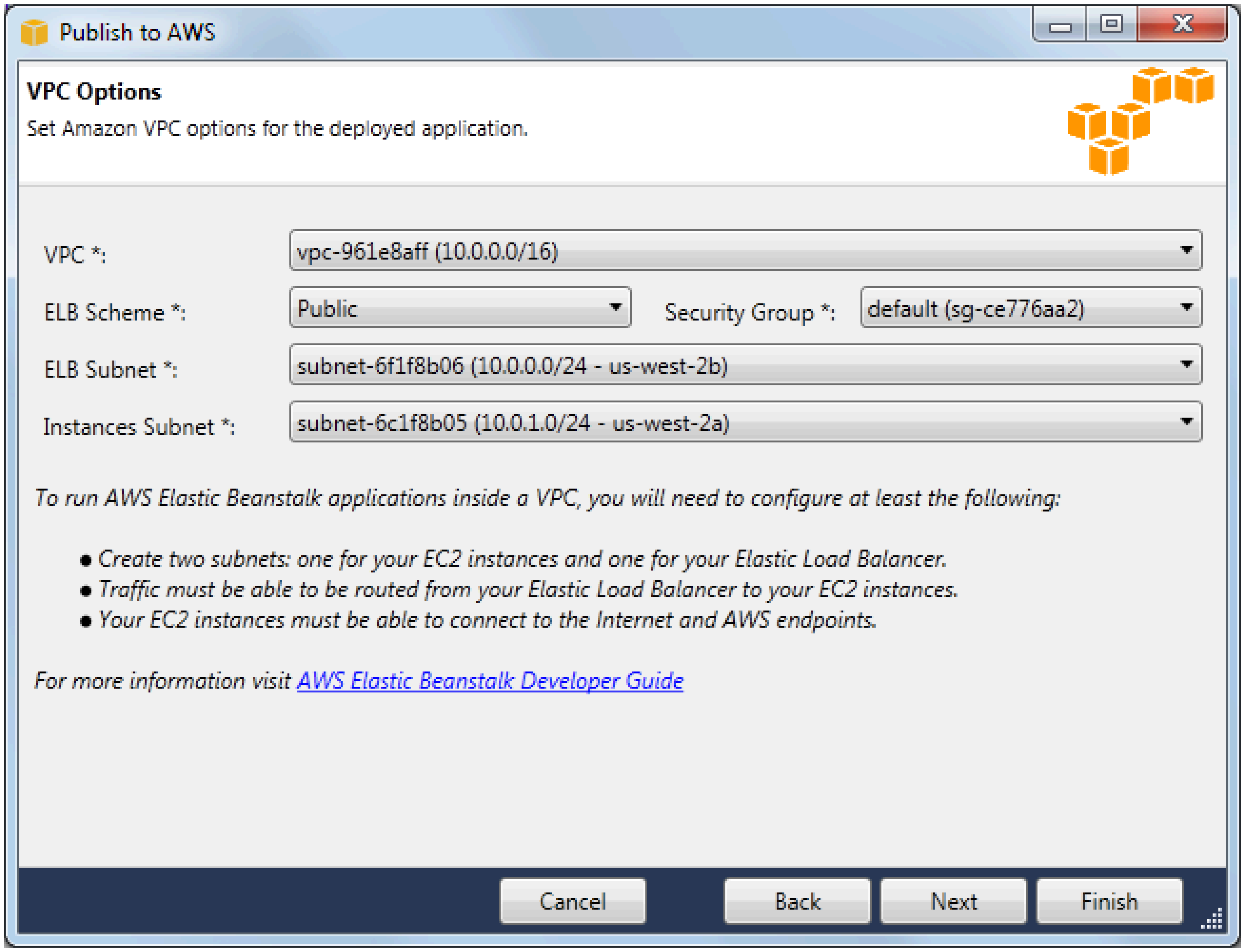
IAM Role: Use the default role (aws-elasticbeanstalk-ec2-role)

If you choose not to use the default role, you must grant the relevant permissions to Elastic Beanstalk. See AWS Elastic Beanstalk Developer Guide for more details.

☐ Launch into VPC *If you elect to launch instances in a VPC, the next page will enable you to customize the VPC settings.*

f. Click **Next**.

6. If you selected to launch your environment inside a VPC, the **VPC Options** page appears; otherwise, the **Additional Options** page appears. Here you'll configure your VPC options.



VPC Options

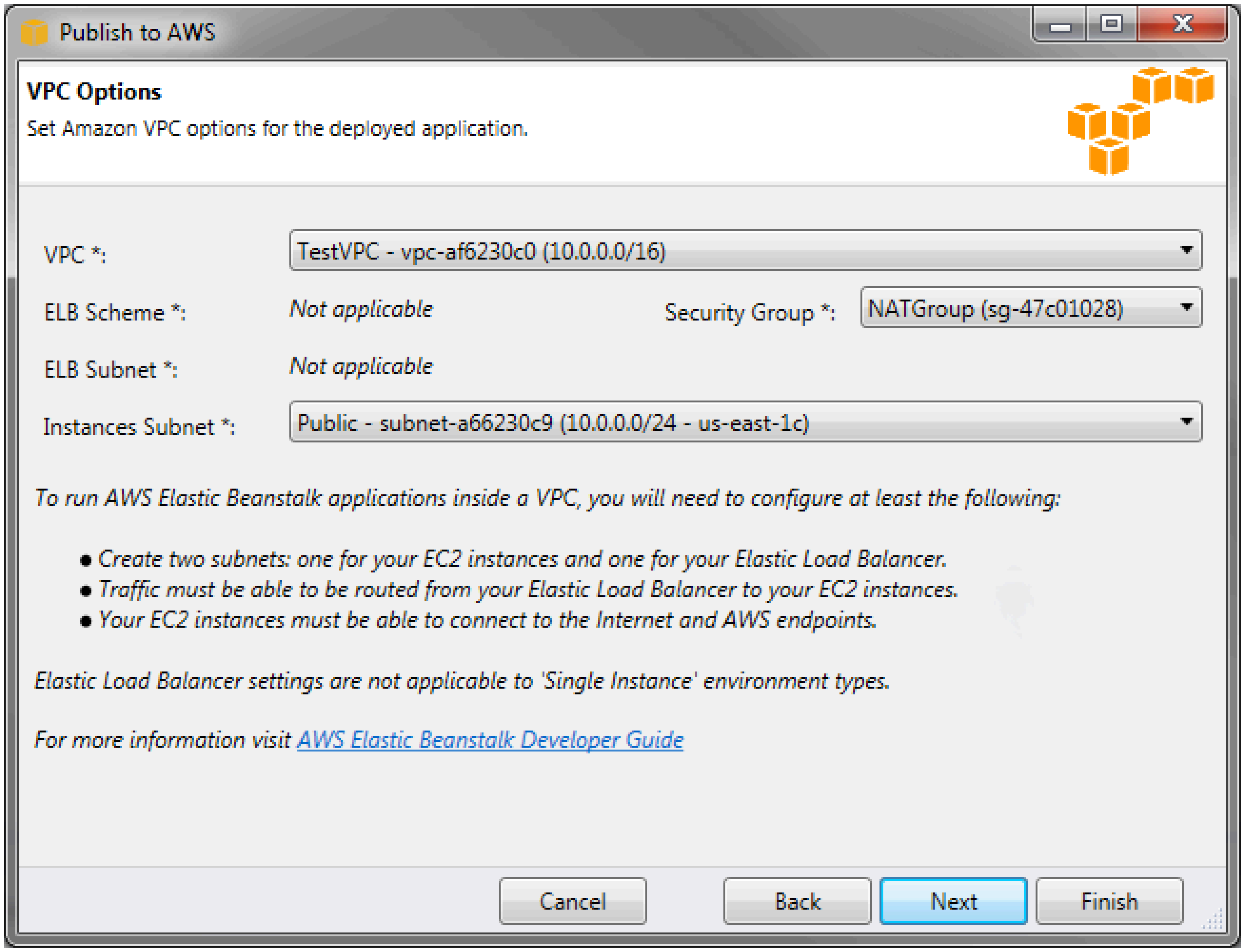
Set Amazon VPC options for the deployed application.

VPC *:	vpc-961e8aff (10.0.0.0/16)
ELB Scheme *:	Public
Security Group *:	default (sg-ce776aa2)
ELB Subnet *:	subnet-6f1f8b06 (10.0.0.0/24 - us-west-2b)
Instances Subnet *:	subnet-6c1f8b05 (10.0.1.0/24 - us-west-2a)

To run AWS Elastic Beanstalk applications inside a VPC, you will need to configure at least the following:

- *Create two subnets: one for your EC2 instances and one for your Elastic Load Balancer.*
- *Traffic must be able to be routed from your Elastic Load Balancer to your EC2 instances.*
- *Your EC2 instances must be able to connect to the Internet and AWS endpoints.*

For more information visit AWS Elastic Beanstalk Developer Guide



Publish to AWS

VPC Options

Set Amazon VPC options for the deployed application.

VPC *: TestVPC - vpc-af6230c0 (10.0.0.0/16)

ELB Scheme *: *Not applicable*

Security Group *: NATGroup (sg-47c01028)

ELB Subnet *: *Not applicable*

Instances Subnet *: Public - subnet-a66230c9 (10.0.0.0/24 - us-east-1c)

To run AWS Elastic Beanstalk applications inside a VPC, you will need to configure at least the following:

- *Create two subnets: one for your EC2 instances and one for your Elastic Load Balancer.*
- *Traffic must be able to be routed from your Elastic Load Balancer to your EC2 instances.*
- *Your EC2 instances must be able to connect to the Internet and AWS endpoints.*

Elastic Load Balancer settings are not applicable to 'Single Instance' environment types.

For more information visit AWS Elastic Beanstalk Developer Guide

a. Select the VPC ID of the VPC in which you would like to launch your environment.

b. For a load-balanced, scalable environment, select **private** for **ELB Scheme** if you do not want your elastic load balancer to be available to the Internet.

   For a single-instance environment, this option is not applicable because the environment doesn't have a load balancer. For more information, see Environment types.

c. For a load-balanced, scalable environment, select the subnets for the elastic load balancer and the EC2 instances. If you created public and private subnets, make sure the elastic load balancer and the EC2 instances are associated with the correct subnet. By default, Amazon VPC creates a default public subnet using 10.0.0.0/24 and a private subnet using 10.0.1.0/24. You can view your existing subnets in the Amazon VPC console at https://console.aws.amazon.com/vpc/.

For a single-instance environment, your VPC only needs a public subnet for the instance. Selecting a subnet for the load balancer is not applicable because the environment doesn't have a load balancer. For more information, see Environment types.

d. For a load-balanced, scalable environment, select the security group you created for your instances, if applicable.

   For a single-instance environment, you don't need a NAT device. Select the default security group. Elastic Beanstalk assigns an Elastic IP address to the instance that lets the instance access the Internet.

e. Click **Next**.

7. On the **Application Options** page, configure your application options.

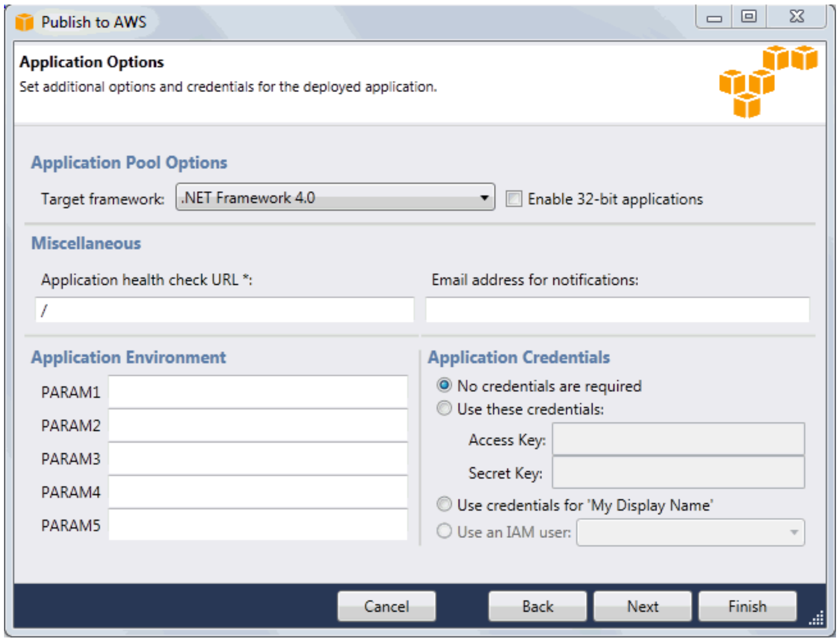
   a. For Target framework, select **.NET Framework 4.0**.

   b. Elastic Load Balancing uses a health check to determine whether the Amazon EC2 instances running your application are healthy. The health check determines an instance's health status by probing a specified URL at a set interval. You can override the default URL to match an existing resource in your application (e.g., `/myapp/index.aspx`) by entering it in the **Application health check URL** box. For more information about application health checks, see Health check.

   c. Type an email address if you want to receive Amazon Simple Notification Service (Amazon SNS) notifications of important events affecting your application.

   d. The **Application Environment** section lets you specify environment variables on the Amazon EC2 instances that are running your application. This setting enables greater portability by eliminating the need to recompile your source code as you move between environments.

   e. Select the application credentials option you want to use to deploy your application.



Application Options

Set additional options and credentials for the deployed application.

Application Pool Options

Target framework:	.NET Framework 4.0
Enable 32-bit applications	☐

Miscellaneous

Application health check URL *:	/
Email address for notifications:	

Application Environment

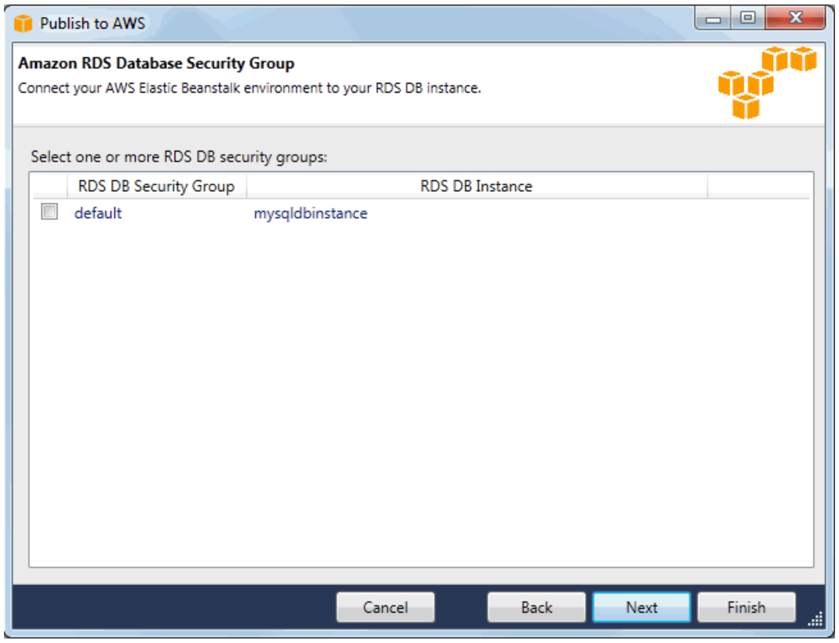
PARAM1	
PARAM2	
PARAM3	
PARAM4	
PARAM5	

Application Credentials

- [✓] No credentials are required
- Use these credentials:
  - Access Key:
  - Secret Key:
- Use credentials for 'My Display Name'
- Use an IAM user:

f. Click **Next**.

8. If you have previously set up an Amazon RDS database, the **Amazon RDS DB Security Group** page appears. If you want to connect your Elastic Beanstalk environment to your Amazon RDS DB Instance, then select one or more security groups. Otherwise, go on to the next step. When you're ready, click **Next**.



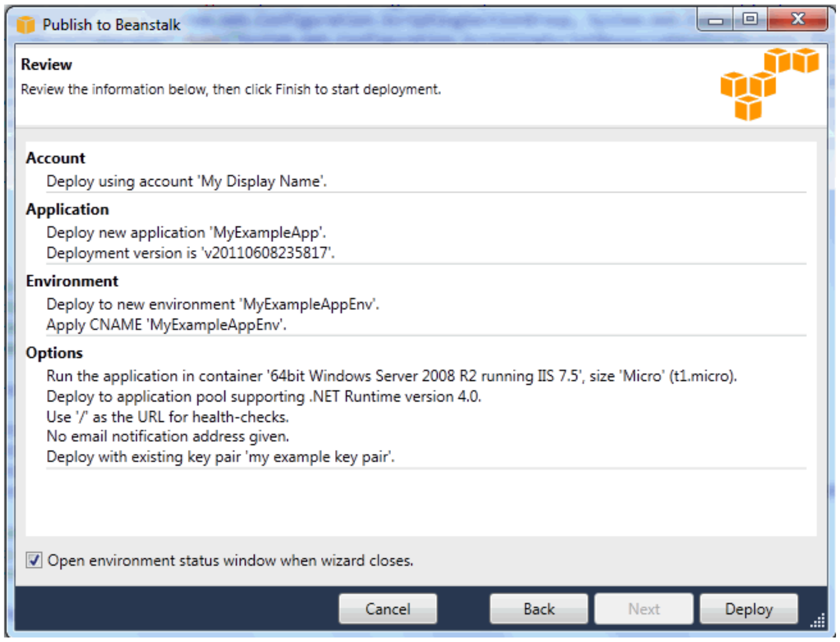
Amazon RDS Database Security Group

Connect your AWS Elastic Beanstalk environment to your RDS DB instance.

Select one or more RDS DB security groups:

	RDS DB Security Group	RDS DB Instance
☐	default	mysqldbinstance

9. Review your deployment options. If everything is as you want, click **Deploy**.



Publish to Beanstalk

Review

Review the information below, then click Finish to start deployment.

Account
Deploy using account 'My Display Name'.

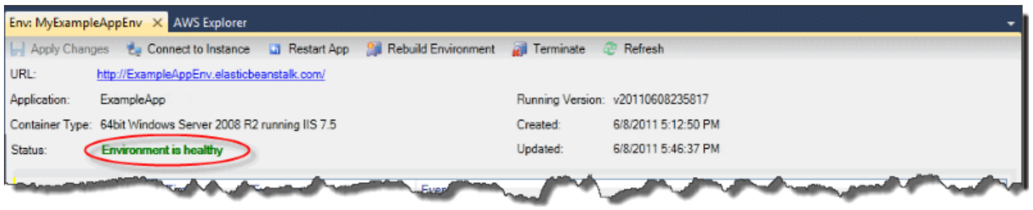
Application
Deploy new application 'MyExampleApp'.
Deployment version is 'v20110608235817'.

Environment
Deploy to new environment 'MyExampleAppEnv'.
Apply CNAME 'MyExampleAppEnv'.

Options
Run the application in container '64bit Windows Server 2008 R2 running IIS 7.5', size 'Micro' (t1.micro).
Deploy to application pool supporting .NET Runtime version 4.0.
Use '/' as the URL for health-checks.
No email notification address given.
Deploy with existing key pair 'my example key pair'.

☑ Open environment status window when wizard closes.

Your ASP.NET project will be exported as a web deploy file, uploaded to Amazon S3, and registered as a new application version with Elastic Beanstalk. The Elastic Beanstalk deployment feature will monitor your environment until it becomes available with the newly deployed code. On the env:<environment name> tab, you will see status for your environment.



URL: http://ExampleAppEnv.elasticbeanstalk.com/
Application: ExampleApp
Container Type: 64bit Windows Server 2008 R2 running IIS 7.5
Status: Environment is healthy
Running Version: v20110608235817
Created: 6/8/2011 5:12:50 PM
Updated: 6/8/2011 5:46:37 PM

Terminating an environment

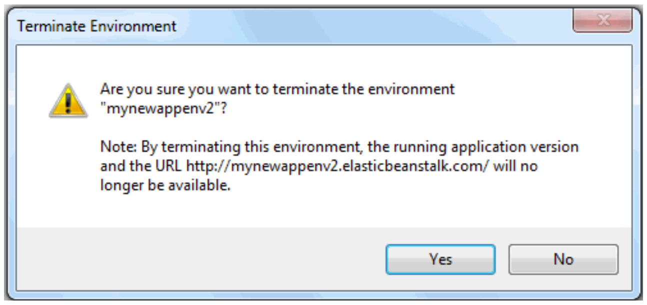
To avoid incurring charges for unused AWS resources, you can terminate a running environment using the AWS Toolkit for Visual Studio.

Note

You can always launch a new environment using the same version later.

To terminate an environment

1. Expand the Elastic Beanstalk node and the application node in **AWS Explorer**. Right-click your application environment and select **Terminate Environment**.
2. When prompted, click **Yes** to confirm that you want to terminate the environment. It will take a few minutes for Elastic Beanstalk to terminate the AWS resources running in the environment.



Are you sure you want to terminate the environment "mynewappenv2"?

Note: By terminating this environment, the running application version and the URL http://mynewappenv2.elasticbeanstalk.com/ will no longer be available.

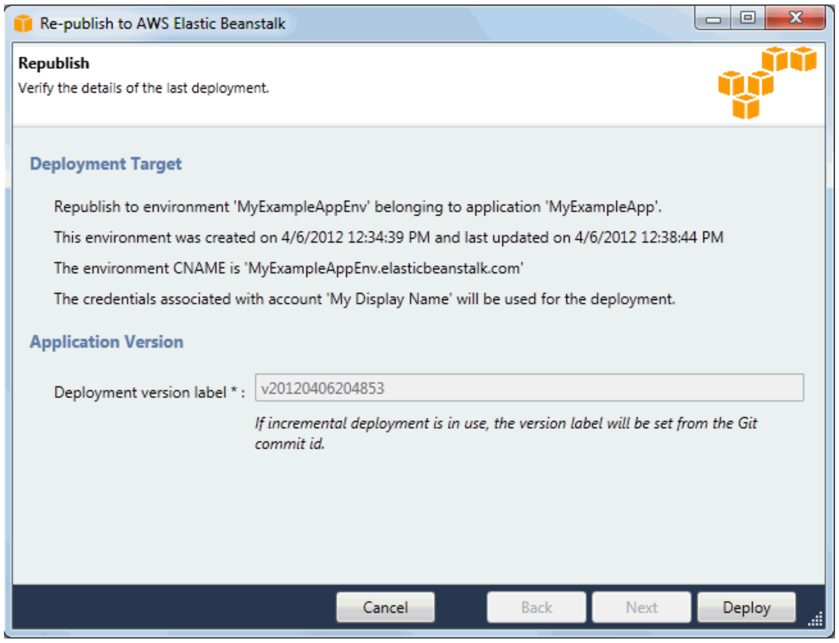
> **Note**
>
> When you terminate your environment, the CNAME associated with the terminated environment becomes available for anyone to use.

Deploying to your environment

Now that you have tested your application, it is easy to edit and redeploy your application and see the results in moments.

To edit and redeploy your ASP.NET web application

1. In **Solution Explorer**, right-click your application, and then click **Republish to Environment <*`your environment name`*>**. The **Re-publish to AWS Elastic Beanstalk** wizard opens.



Republish

Verify the details of the last deployment.

Deployment Target

Republish to environment 'MyExampleAppEnv' belonging to application 'MyExampleApp'.

This environment was created on 4/6/2012 12:34:39 PM and last updated on 4/6/2012 12:38:44 PM

The environment CNAME is 'MyExampleAppEnv.elasticbeanstalk.com'

The credentials associated with account 'My Display Name' will be used for the deployment.

Application Version

Deployment version label * : v20120406204853

If incremental deployment is in use, the version label will be set from the Git commit id.

2. Review your deployment details and click **Deploy**.

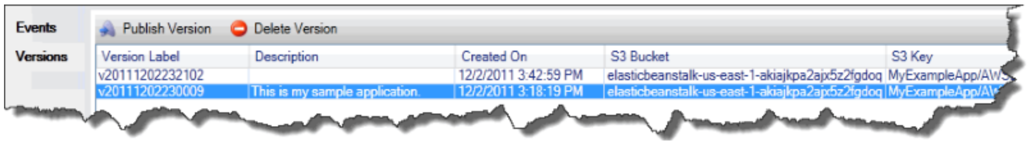
> **Note**
>
> If you want to change any of your settings, you can click **Cancel** and use the **Publish to AWS** wizard instead. For instructions, see Create an Elastic Beanstalk environment.

Your updated ASP.NET web project will be exported as a web deploy file with the new version label, uploaded to Amazon S3, and registered as a new application version with Elastic Beanstalk. The Elastic Beanstalk deployment feature monitors your existing environment until it becomes available with the newly deployed code. On the **env:<*environment name*>** tab, you will see the status of your environment.

You can also deploy an existing application to an existing environment if, for instance, you need to roll back to a previous application version.

To deploy an application version to an existing environment

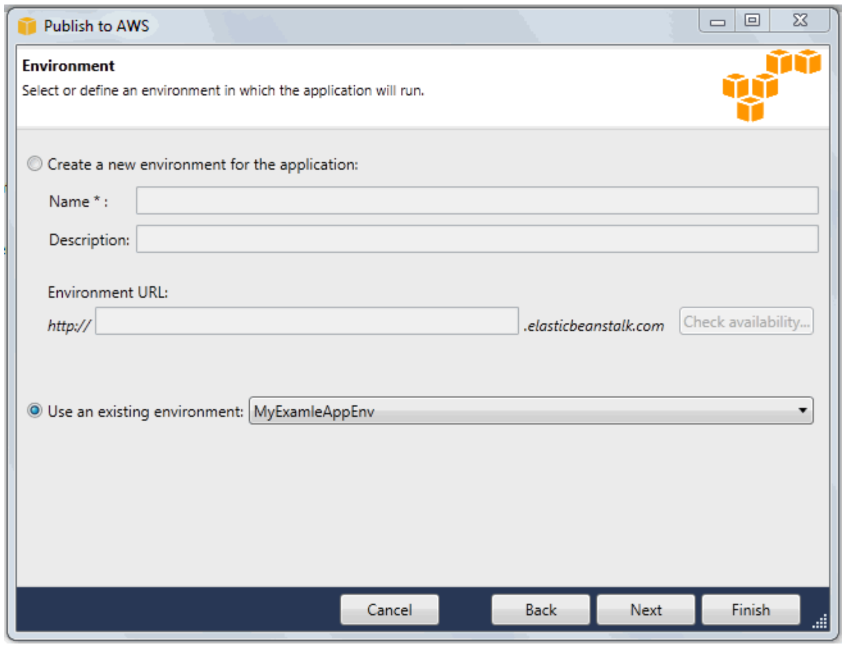
1. Right-click your Elastic Beanstalk application by expanding the Elastic Beanstalk node in **AWS Explorer**. Select **View Status**.
2. In the **App: <*application name*>** tab, click **Versions**.



Version Label	Description	Created On	S3 Bucket	S3 Key
v20111202232102		12/2/2011 3:42:59 PM	elasticbeanstalk-us-east-1-akiajkpa2ajx5z2fgdoq	MyExampleApp/AWS
v20111202230009	This is my sample application.	12/2/2011 3:18:19 PM	elasticbeanstalk-us-east-1-akiajkpa2ajx5z2fgdoq	MyExampleApp/AV

3. Click the application version you want to deploy and click **Publish Version**.

4. In the **Publish Application Version** wizard, click **Next**.

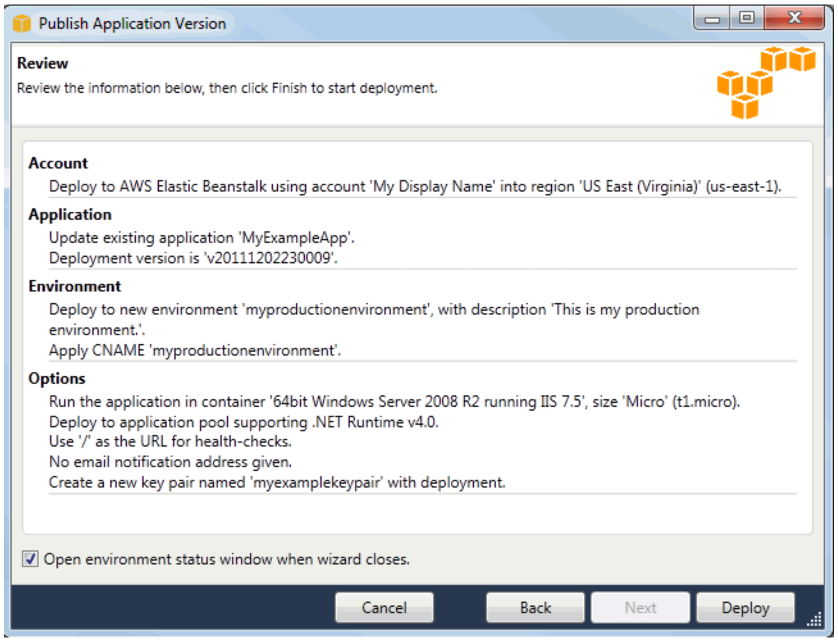


Environment
Select or define an environment in which the application will run.

( ) Create a new environment for the application:
Name * :
Description:
Environment URL:
http:// .elasticbeanstalk.com

(●) Use an existing environment: MyExamleAppEnv

5. Review your deployment options, and click **Deploy**.



Review
Review the information below, then click Finish to start deployment.

Account
Deploy to AWS Elastic Beanstalk using account 'My Display Name' into region 'US East (Virginia)' (us-east-1).

Application
Update existing application 'MyExampleApp'.
Deployment version is 'v20111202230009'.

Environment
Deploy to new environment 'myproductionenvironment', with description 'This is my production environment.'.
Apply CNAME 'myproductionenvironment'.

Options
Run the application in container '64bit Windows Server 2008 R2 running IIS 7.5', size 'Micro' (t1.micro).
Deploy to application pool supporting .NET Runtime v4.0.
Use '/' as the URL for health-checks.
No email notification address given.
Create a new key pair named 'myexamplekeypair' with deployment.

☑ Open environment status window when wizard closes.

Your ASP.NET project will be exported as a web deploy file and uploaded to Amazon S3. The Elastic Beanstalk deployment feature will monitor your environment until it becomes available with the newly deployed code. On the **env:<*`environment name`*>** tab, you will see status for your environment.

Managing your Elastic Beanstalk application environments

With the AWS Toolkit for Visual Studio and the AWS Management Console, you can change the provisioning and configuration of the AWS resources used by your application environments. For information on how to manage your application environments using the AWS Management Console, see Creating environments in Elastic Beanstalk. This section discusses the specific service settings you can edit in the AWS Toolkit for Visual Studio as part of your application environment configuration.

Changing environment configurations settings

When you deploy your application, Elastic Beanstalk configures a number of AWS cloud computing services. You can control how these individual services are configured using the AWS Toolkit for Visual Studio.

To edit an application's environment settings

- Expand the Elastic Beanstalk node and your application node. Then right-click your Elastic Beanstalk environment in **AWS Explorer**. Select **View Status**.

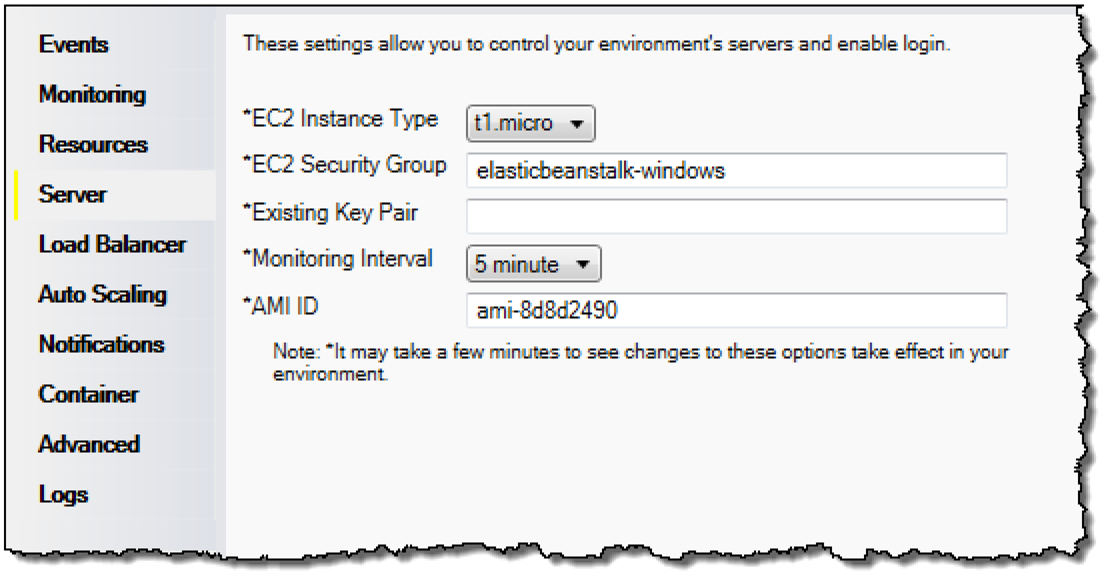
  You can now configure settings for the following:

  - Server
  - Load balancing
  - Autoscaling
  - Notifications
  - Environment properties

Configuring EC2 server instances using the AWS toolkit for Visual Studio

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that you use to launch and manage server instances in Amazon's data centers. You can use Amazon EC2 server instances at any time, for as long as you need, and for any legal purpose. Instances are available in different sizes and configurations. For more information, go to [Amazon EC2](#).

You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration with the **Server** tab inside your application environment tab in the AWS Toolkit for Visual Studio.



These settings allow you to control your environment's servers and enable login.

*EC2 Instance Type	t1.micro
*EC2 Security Group	elasticbeanstalk-windows
*Existing Key Pair	
*Monitoring Interval	5 minute
*AMI ID	ami-8d8d2490

Note: *It may take a few minutes to see changes to these options take effect in your environment.

Amazon EC2 instance types

Instance type displays the instance types available to your Elastic Beanstalk application. Change the instance type to select a server with the characteristics (including memory size and CPU power) that are most appropriate to your application. For example, applications with intensive and long-running operations can require more CPU or memory.

For more information about the Amazon EC2 instance types available for your Elastic Beanstalk application, see [Instance Types](#) in the *Amazon Elastic Compute Cloud User Guide*.

Amazon EC2 security groups

You can control access to your Elastic Beanstalk application using an *Amazon EC2 Security Group*. A security group defines firewall rules for your instances. These rules specify which ingress (i.e., incoming) network traffic should be delivered to your instance. All other ingress traffic will be discarded. You can modify rules for a group at any time. The new rules are automatically enforced for all running instances and instances launched in the future.

You can set up your Amazon EC2 security groups using the AWS Management Console or by using the AWS Toolkit for Visual Studio. You can specify which Amazon EC2 Security Groups control access to your Elastic Beanstalk application by entering the names of one or more Amazon EC2 security group names (delimited by commas) into the **EC2 Security Groups** text box.

Note

Make sure port 80 (HTTP) is accessible from 0.0.0.0/0 as the source CIDR range if you want to enable health checks for your application. For more information about health checks, see Health checks.

To create a security group using the AWS toolkit for Visual Studio

1. In Visual Studio, in **AWS Explorer**, expand the **Amazon EC2** node, and then double-click **Security Groups**.
2. Click **Create Security Group**, and enter a name and description for your security group.
3. Click **OK**.

For more information on Amazon EC2 Security Groups, see Using Security Groups in the *Amazon Elastic Compute Cloud User Guide*.

Amazon EC2 key pairs

You can securely log in to the Amazon EC2 instances provisioned for your Elastic Beanstalk application with an Amazon EC2 key pair.

Important

You must create an Amazon EC2 key pair and configure your Elastic Beanstalk–provisioned Amazon EC2 instances to use the Amazon EC2 key pair before you can access your Elastic Beanstalk–provisioned Amazon EC2 instances. You can create your key pair using the **Publish to AWS** wizard inside the AWS Toolkit for Visual Studio when you deploy your application to Elastic Beanstalk. If you want to create additional key pairs using the Toolkit, follow the steps below. Alternatively, you can set up your Amazon EC2 key pairs using the AWS Management Console. For instructions on creating a key pair for Amazon EC2, see the Amazon Elastic Compute Cloud Getting Started Guide.

The **Existing Key Pair** text box lets you specify the name of an Amazon EC2 key pair you can use to securely log in to the Amazon EC2 instances running your Elastic Beanstalk application.

To specify the name of an Amazon EC2 key pair

1. Expand the **Amazon EC2** node and double-click **Key Pairs**.
2. Click **Create Key Pair** and enter the key pair name.
3. Click **OK**.

For more information about Amazon EC2 key pairs, go to Using Amazon EC2 Credentials in the *Amazon Elastic Compute Cloud User Guide*. For more information about connecting to Amazon EC2 instances, see Listing and connecting to server instances.

Monitoring interval

By default, only basic Amazon CloudWatch metrics are enabled. They return data in five-minute periods. You can enable more granular one-minute CloudWatch metrics by selecting **1 minute** for the **Monitoring Interval** in the **Server** section of the **Configuration** tab for your environment in the AWS Toolkit for Eclipse.

Note

Amazon CloudWatch service charges can apply for one-minute interval metrics. See Amazon CloudWatch for more information.

Custom AMI ID

You can override the default AMI used for your Amazon EC2 instances with your own custom AMI by entering the identifier of your custom AMI into the **Custom AMI ID** box in the **Server** section of the **Configuration** tab for your environment in the AWS Toolkit for Eclipse.

Important

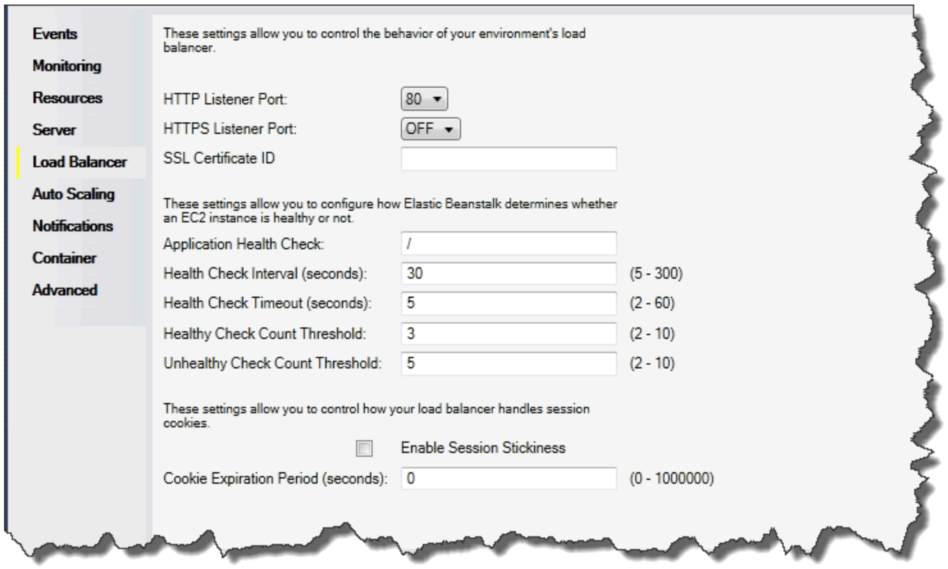
Using your own AMI is an advanced task that you should do with care. If you need a custom AMI, we recommend you start with the default Elastic Beanstalk AMI and then modify it. To be considered healthy, Elastic Beanstalk expects Amazon EC2 instances to meet a set of requirements, including having a running host manager. If these requirements are not met, your environment might not work properly.

Configuring Elastic Load Balancing using the AWS toolkit for Visual Studio

Elastic Load Balancing is an Amazon web service that helps you improve the availability and scalability of your application. This service makes it easy for you to distribute application loads between two or more Amazon EC2 instances. Elastic Load Balancing enables availability through redundancy and supports traffic growth for your application.

Elastic Load Balancing lets you automatically distribute and balance the incoming application traffic among all the instances you are running. The service also makes it easy to add new instances when you need to increase the capacity of your application.

Elastic Beanstalk automatically provisions Elastic Load Balancing when you deploy an application. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration with the **Load Balancer** tab inside your application environment tab in AWS Toolkit for Visual Studio.



These settings allow you to control the behavior of your environment's load balancer.

HTTP Listener Port:	80	
HTTPS Listener Port:	OFF	
SSL Certificate ID		

These settings allow you to configure how Elastic Beanstalk determines whether an EC2 instance is healthy or not.

Application Health Check:	/	
Health Check Interval (seconds):	30	(5 - 300)
Health Check Timeout (seconds):	5	(2 - 60)
Healthy Check Count Threshold:	3	(2 - 10)
Unhealthy Check Count Threshold:	5	(2 - 10)

These settings allow you to control how your load balancer handles session cookies.

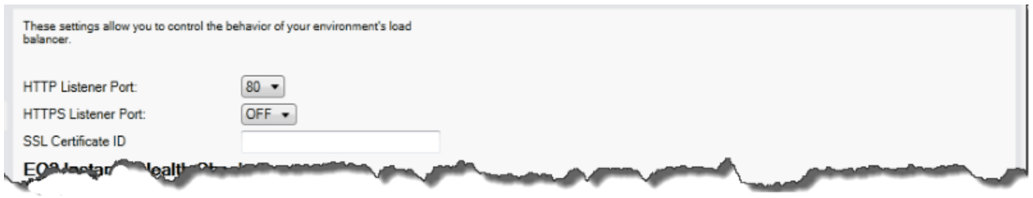
[ ] Enable Session Stickiness

Cookie Expiration Period (seconds):	0	(0 - 1000000)

The following sections describe the Elastic Load Balancing parameters you can configure for your application.

Ports

The load balancer provisioned to handle requests for your Elastic Beanstalk application sends requests to the Amazon EC2 instances that are running your application. The provisioned load balancer can listen for requests on HTTP and HTTPS ports and route requests to the Amazon EC2 instances in your AWS Elastic Beanstalk application. By default, the load balancer handles requests on the HTTP port. At least one of the ports (either HTTP or HTTPS) must be turned on.



These settings allow you to control the behavior of your environment's load balancer.

HTTP Listener Port: 80

HTTPS Listener Port: OFF

SSL Certificate ID

> **Important**
>
> Make sure that the port you specified is not locked down; otherwise, users will not be able to connect to your Elastic Beanstalk application.

Controlling the HTTP port

To turn off the HTTP port, select **OFF** for **HTTP Listener Port**. To turn on the HTTP port, you select an HTTP port (for example, **80**) from the list.

> **Note**
>
> To access your environment using a port other than the default port 80, such as port 8080, add a listener to the existing load balancer and configure the new listener to listen on that port.
> For example, using the AWS CLI for Classic load balancers, type the following command, replacing *LOAD_BALANCER_NAME* with the name of your load balancer for Elastic Beanstalk.
>
> ```
> aws elb create-load-balancer-listeners --load-balancer-name LOAD_BALANCER_NAME
>  --listeners "Protocol=HTTP, LoadBalancerPort=8080, InstanceProtocol=HTTP,
>  InstancePort=80"
> ```
>
> For example, using the AWS CLI for Application Load Balancers, type the following command, replacing *LOAD_BALANCER_ARN* with the ARN of your load balancer for Elastic Beanstalk.
>
> ```
> aws elbv2 create-listener --load-balancer-arn LOAD_BALANCER_ARN --protocol HTTP
>  --port 8080
> ```

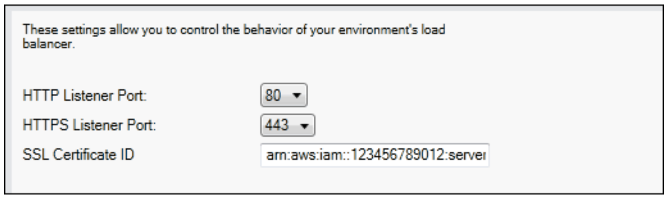
If you want Elastic Beanstalk to monitor your environment, do not remove the listener on port 80.

Controlling the HTTPS port

Elastic Load Balancing supports the HTTPS/TLS protocol to enable traffic encryption for client connections to the load balancer. Connections from the load balancer to the EC2 instances use plaintext encryption. By default, the HTTPS port is turned off.

To turn on the HTTPS port

1. Create a new certificate using AWS Certificate Manager (ACM) or upload a certificate and key to AWS Identity and Access Management (IAM). For more information about requesting an ACM certificate, see Request a Certificate in the *AWS Certificate Manager User Guide*. For more information about importing third-party certificates into ACM, see Importing Certificates in the *AWS Certificate Manager User Guide*. If ACM is not available in your region, use AWS Identity and Access Management (IAM) to upload a third-party certificate. The ACM and IAM services store the certificate and provide an Amazon Resource Name (ARN) for the SSL certificate. For more information about creating and uploading certificates to IAM, see Working with Server Certificates in *IAM User Guide*.
2. Specify the HTTPS port by selecting a port for **HTTPS Listener Port**.



These settings allow you to control the behavior of your environment's load balancer.

HTTP Listener Port:	80
HTTPS Listener Port:	443
SSL Certificate ID	arn:aws:iam::123456789012:server

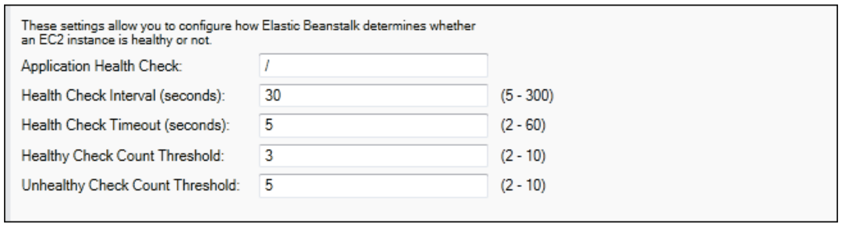
3. For **SSL Certificate ID**, enter the Amazon Resources Name (ARN) of your SSL certificate. For example, **`arn:aws:iam::123456789012:server-certificate/abc/certs/build`** or **`arn:aws:acm:us-east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678`**. Use the SSL certificate that you created or uploaded in step 1.

To turn off the HTTPS port, select **OFF** for **HTTPS Listener Port**.

Health checks

The health check definition includes a URL to be queried for instance health. By default, Elastic Beanstalk uses TCP:80 for nonlegacy containers and HTTP:80 for legacy containers. You can override the default URL to match an existing resource in your application (e.g., `/myapp/default.aspx`) by entering it in the **Application Health Check URL** box. If you override the default URL, then Elastic Beanstalk uses HTTP to query the resource. To check if you are using a legacy container type, see the section called "Why are some platform versions marked legacy?"

You can control the settings for the health check using the **EC2 Instance Health Check** section of the **Load Balancing** panel.



These settings allow you to configure how Elastic Beanstalk determines whether an EC2 instance is healthy or not.

Application Health Check:	/	
Health Check Interval (seconds):	30	(5 - 300)
Health Check Timeout (seconds):	5	(2 - 60)
Healthy Check Count Threshold:	3	(2 - 10)
Unhealthy Check Count Threshold:	5	(2 - 10)

The health check definition includes a URL to be queried for instance health. Override the default URL to match an existing resource in your application (e.g., `/myapp/index.jsp`) by entering it in the **Application Health Check URL** box.

The following list describes the health check parameters you can set for your application.

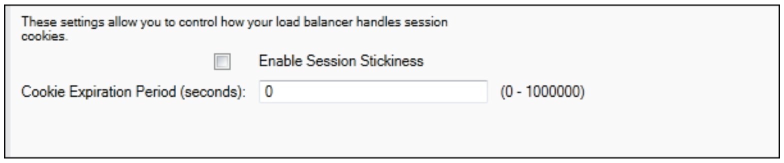
- For **Health Check Interval (seconds)**, enter the number of seconds Elastic Load Balancing waits between health checks for your application's Amazon EC2 instances.
- For **Health Check Timeout (seconds)**, specify the number of seconds Elastic Load Balancing waits for a response before it considers the instance unresponsive.
- For **Healthy Check Count Threshold** and **Unhealthy Check Count Threshold**, specify the number of consecutive successful or unsuccessful URL probes before Elastic Load Balancing changes the instance health status. For example, specifying **5** for **Unhealthy Check Count Threshold** means that the URL would have to return an error message or timeout five consecutive times before Elastic Load Balancing considers the health check failed.

Sessions

By default, a load balancer routes each request independently to the server instance with the smallest load. By comparison, a sticky session binds a user's session to a specific server instance so that all requests coming from the user during the session are sent to the same server instance.

Elastic Beanstalk uses load balancer–generated HTTP cookies when sticky sessions are enabled for an application. The load balancer uses a special load balancer–generated cookie to track the application instance for each request. When the load balancer receives a request, it first checks to see if this cookie is present in the request. If so, the request is sent to the application instance specified in the cookie. If there is no cookie, the load balancer chooses an application instance based on the existing load balancing algorithm. A cookie is inserted into the response for binding subsequent requests from the same user to that application instance. The policy configuration defines a cookie expiry, which establishes the duration of validity for each cookie.

You can use the **Sessions** section on the **Load Balancer** tab to specify whether or not the load balancer for your application allows session stickiness.


These settings allow you to control how your load balancer handles session cookies.

☐ Enable Session Stickiness

Cookie Expiration Period (seconds): 0 (0 - 1000000)

For more information on Elastic Load Balancing, go to the Elastic Load Balancing Developer Guide.

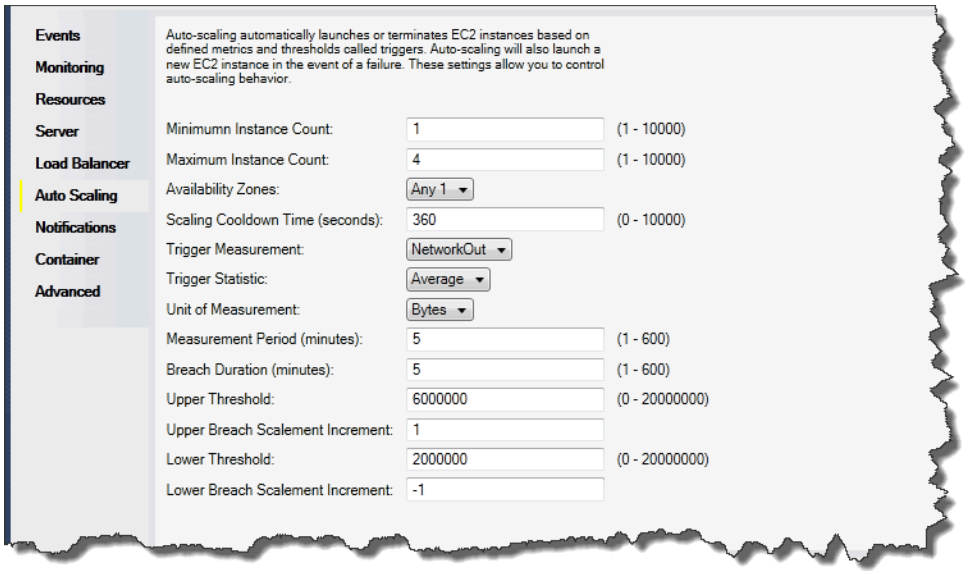
Configuring Auto Scaling using the AWS toolkit for Visual Studio

Amazon EC2 Auto Scaling is an Amazon web service designed to automatically launch or terminate Amazon EC2 instances based on user-defined triggers. Users can set up *Auto Scaling groups* and associate *triggers* with these groups to automatically scale computing resources based on metrics such as bandwidth usage or CPU utilization. Amazon EC2 Auto Scaling works with Amazon CloudWatch to retrieve metrics for the server instances running your application.

Amazon EC2 Auto Scaling lets you take a group of Amazon EC2 instances and set various parameters to have this group automatically increase or decrease in number. Amazon EC2 Auto Scaling can add or remove Amazon EC2 instances from that group to help you seamlessly deal with traffic changes to your application.

Amazon EC2 Auto Scaling also monitors the health of each Amazon EC2 instance that it launches. If any instance terminates unexpectedly, Amazon EC2 Auto Scaling detects the termination and launches a replacement instance. This capability enables you to maintain a fixed, desired number of Amazon EC2 instances automatically.

Elastic Beanstalk provisions Amazon EC2 Auto Scaling for your application. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration with the **Auto Scaling** tab inside your application environment tab in the AWS Toolkit for Visual Studio.



Auto-scaling automatically launches or terminates EC2 instances based on defined metrics and thresholds called triggers. Auto-scaling will also launch a new EC2 instance in the event of a failure. These settings allow you to control auto-scaling behavior.

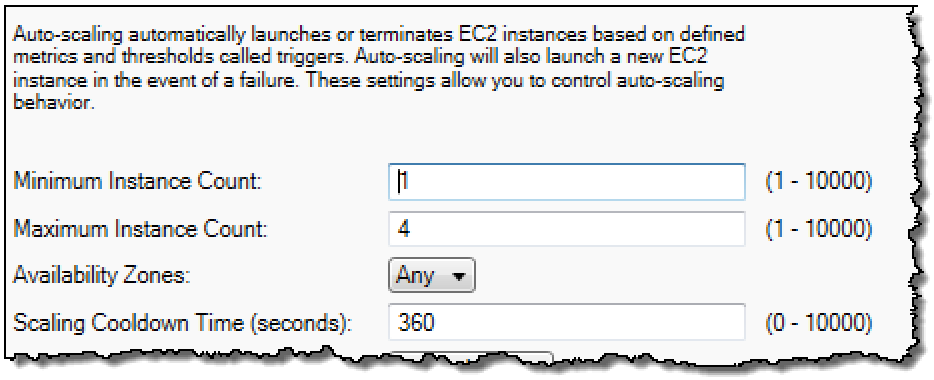
Minimumn Instance Count:	1	(1 - 10000)
Maximum Instance Count:	4	(1 - 10000)
Availability Zones:	Any 1	
Scaling Cooldown Time (seconds):	360	(0 - 10000)
Trigger Measurement:	NetworkOut	
Trigger Statistic:	Average	
Unit of Measurement:	Bytes	
Measurement Period (minutes):	5	(1 - 600)
Breach Duration (minutes):	5	(1 - 600)
Upper Threshold:	6000000	(0 - 20000000)
Upper Breach Scalement Increment:	1	
Lower Threshold:	2000000	(0 - 20000000)
Lower Breach Scalement Increment:	-1	

The following section discusses how to configure Auto Scaling parameters for your application.

Launch the configuration

You can edit the launch configuration to control how your Elastic Beanstalk application provisions Amazon EC2 Auto Scaling resources.

The **Minimum Instance Count** and **Maximum Instance Count** boxes let you specify the minimum and maximum size of the Auto Scaling group that your Elastic Beanstalk application uses.



Auto-scaling automatically launches or terminates EC2 instances based on defined metrics and thresholds called triggers. Auto-scaling will also launch a new EC2 instance in the event of a failure. These settings allow you to control auto-scaling behavior.

Minimum Instance Count:	1	(1 - 10000)
Maximum Instance Count:	4	(1 - 10000)
Availability Zones:	Any	
Scaling Cooldown Time (seconds):	360	(0 - 10000)

Note

To maintain a fixed number of Amazon EC2 instances, set **Minimum Instance Count** and **Maximum Instance Count** to the same value.

The **Availability Zones** box lets you specify the number of Availability Zones you want your Amazon EC2 instances to be in. It is important to set this number if you want to build fault-tolerant applications. If one Availability Zone goes down, your instances will still be running in your other Availability Zones.

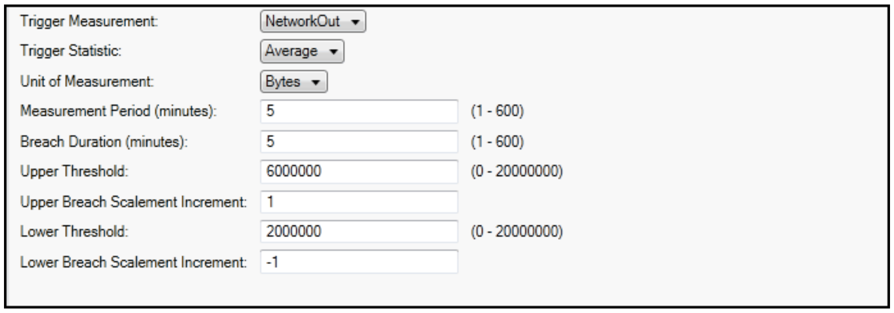
Note

Currently, it is not possible to specify which Availability Zone your instance will be in.

Triggers

A *trigger* is an Amazon EC2 Auto Scaling mechanism that you set to tell the system when you want to increase (*scale out*) the number of instances, and when you want to decrease (*scale in*) the number of instances. You can configure triggers to *fire* on any metric published to Amazon CloudWatch, such as CPU utilization, and determine if the conditions you specified have been met. When the upper or lower thresholds of the conditions you have specified for the metric have been breached for the specified period of time, the trigger launches a long-running process called a *Scaling Activity*.

You can define a scaling trigger for your Elastic Beanstalk application using AWS Toolkit for Visual Studio.



Trigger Measurement:	NetworkOut	
Trigger Statistic:	Average	
Unit of Measurement:	Bytes	
Measurement Period (minutes):	5	(1 - 600)
Breach Duration (minutes):	5	(1 - 600)
Upper Threshold:	6000000	(0 - 20000000)
Upper Breach Scalement Increment:	1	
Lower Threshold:	2000000	(0 - 20000000)
Lower Breach Scalement Increment:	-1	

Amazon EC2 Auto Scaling triggers work by watching a specific Amazon CloudWatch metric for an instance. Triggers include CPU utilization, network traffic, and disk activity. Use the **Trigger Measurement** setting to select a metric for your trigger.

The following list describes the trigger parameters you can configure using the AWS Management Console.

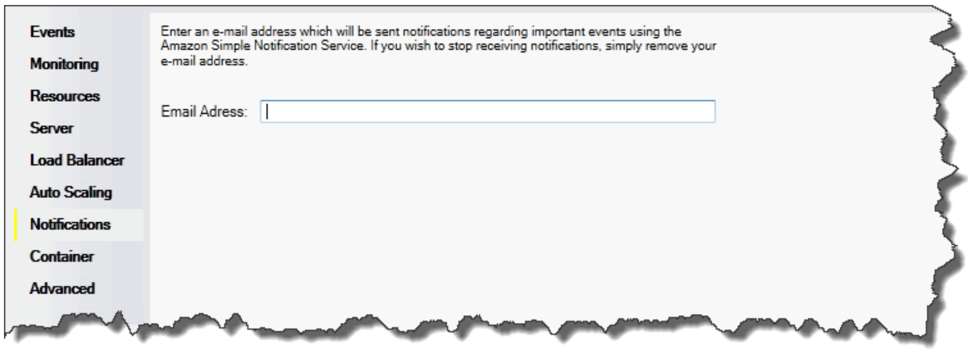
- You can specify which statistic the trigger should use. You can select **Minimum**, **Maximum**, **Sum**, or **Average** for **Trigger Statistic**.

- For **Unit of Measurement**, specify the unit for the trigger measurement.
- The value in the **Measurement Period** box specifies how frequently Amazon CloudWatch measures the metrics for your trigger. The **Breach Duration** is the amount of time a metric can be beyond its defined limit (as specified for the **Upper Threshold** and **Lower Threshold**) before the trigger fires.
- For **Upper Breach Scale Increment** and **Lower Breach Scale Increment**, specify how many Amazon EC2 instances to add or remove when performing a scaling activity.

For more information on Amazon EC2 Auto Scaling, see the *Amazon EC2 Auto Scaling* section on Amazon Elastic Compute Cloud Documentation.

Configuring notifications using AWS toolkit for Visual Studio

Elastic Beanstalk uses the Amazon Simple Notification Service (Amazon SNS) to notify you of important events affecting your application. To enable Amazon SNS notifications, simply enter your email address in the **Email Address** box. To disable these notifications, remove your email address from the box.



Enter an e-mail address which will be sent notifications regarding important events using the Amazon Simple Notification Service. If you wish to stop receiving notifications, simply remove your e-mail address.

Email Adress:

Configuring .NET containers using the AWS toolkit for Visual Studio

The **Container/.NET Options** panel lets you fine-tune the behavior of your Amazon EC2 instances and enable or disable Amazon S3 log rotation. You can use the AWS Toolkit for Visual Studio to configure your container information.

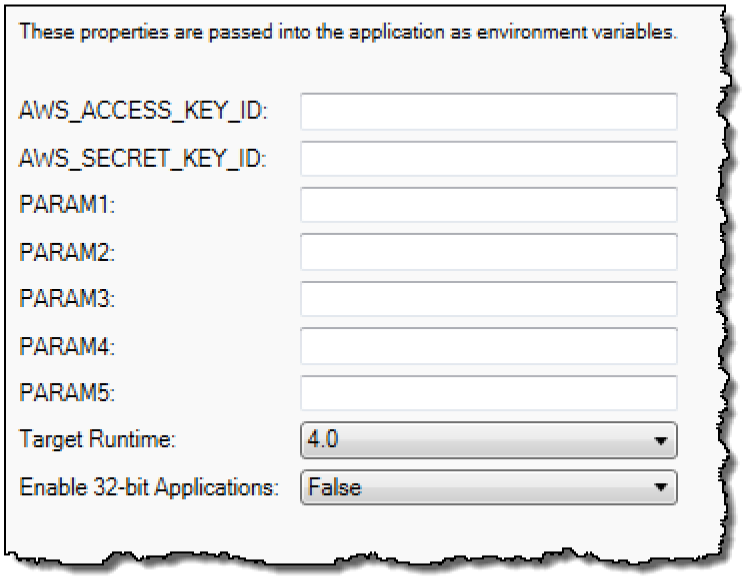
Note

You can modify your configuration settings with zero downtime by swapping the CNAME for your environments. For more information, see Blue/Green deployments with Elastic Beanstalk.

If you want to, you can extend the number of parameters. For information about extending parameters, see Option settings.

To access the Container/.NET options panel for your Elastic Beanstalk application

1. In AWS Toolkit for Visual Studio, expand the Elastic Beanstalk node and your application node.
2. In **AWS Explorer**, double-click your Elastic Beanstalk environment.
3. At the bottom of the **Overview** pane, click the **Configuration** tab.
4. Under **Container**, you can configure container options.



These properties are passed into the application as environment variables.

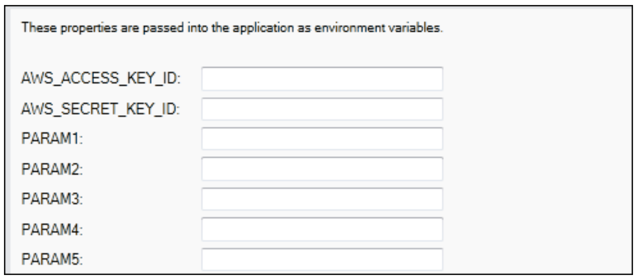
AWS_ACCESS_KEY_ID:	
AWS_SECRET_KEY_ID:	
PARAM1:	
PARAM2:	
PARAM3:	
PARAM4:	
PARAM5:	
Target Runtime:	4.0
Enable 32-bit Applications:	False

.NET container options

You can choose the version of .NET Framework for your application. Choose either 2.0 or 4.0 for **Target runtime**. Select **Enable 32-bit Applications** if you want to enable 32-bit applications.

Application settings

The **Application Settings** section lets you specify environment variables that you can read from your application code.



These properties are passed into the application as environment variables.

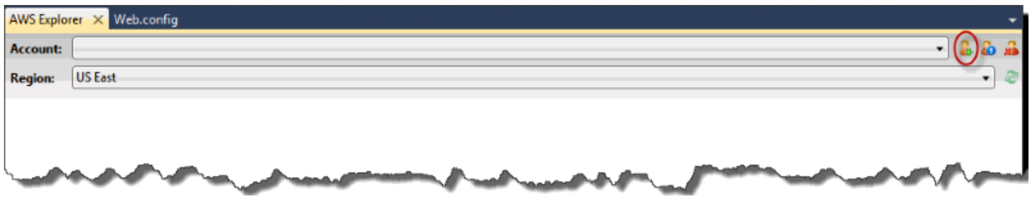
AWS_ACCESS_KEY_ID:	
AWS_SECRET_KEY_ID:	
PARAM1:	
PARAM2:	
PARAM3:	
PARAM4:	
PARAM5:	

Managing accounts

If you want to set up different AWS accounts to perform different tasks, such as testing, staging, and production, you can add, edit, and delete accounts using the AWS Toolkit for Visual Studio.

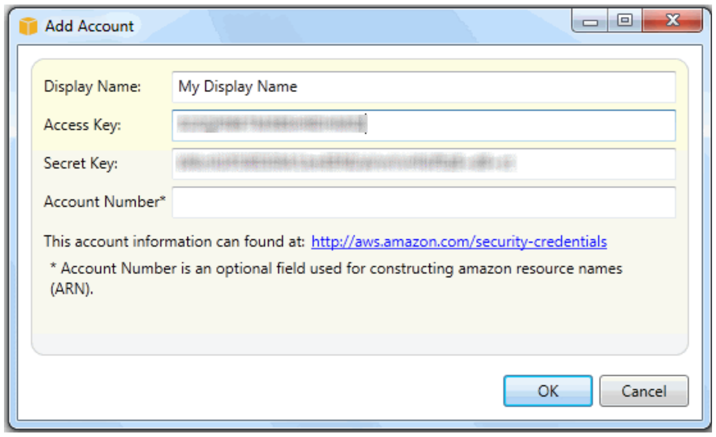
To manage multiple accounts

1. In Visual Studio, on the **View** menu, click **AWS Explorer**.
2. Beside the **Account** list, click the **Add Account** button.



Account:
Region: US East

The **Add Account** dialog box appears.



Display Name: My Display Name
Access Key:
Secret Key:
Account Number*

This account information can found at: http://aws.amazon.com/security-credentials

* Account Number is an optional field used for constructing amazon resource names (ARN).

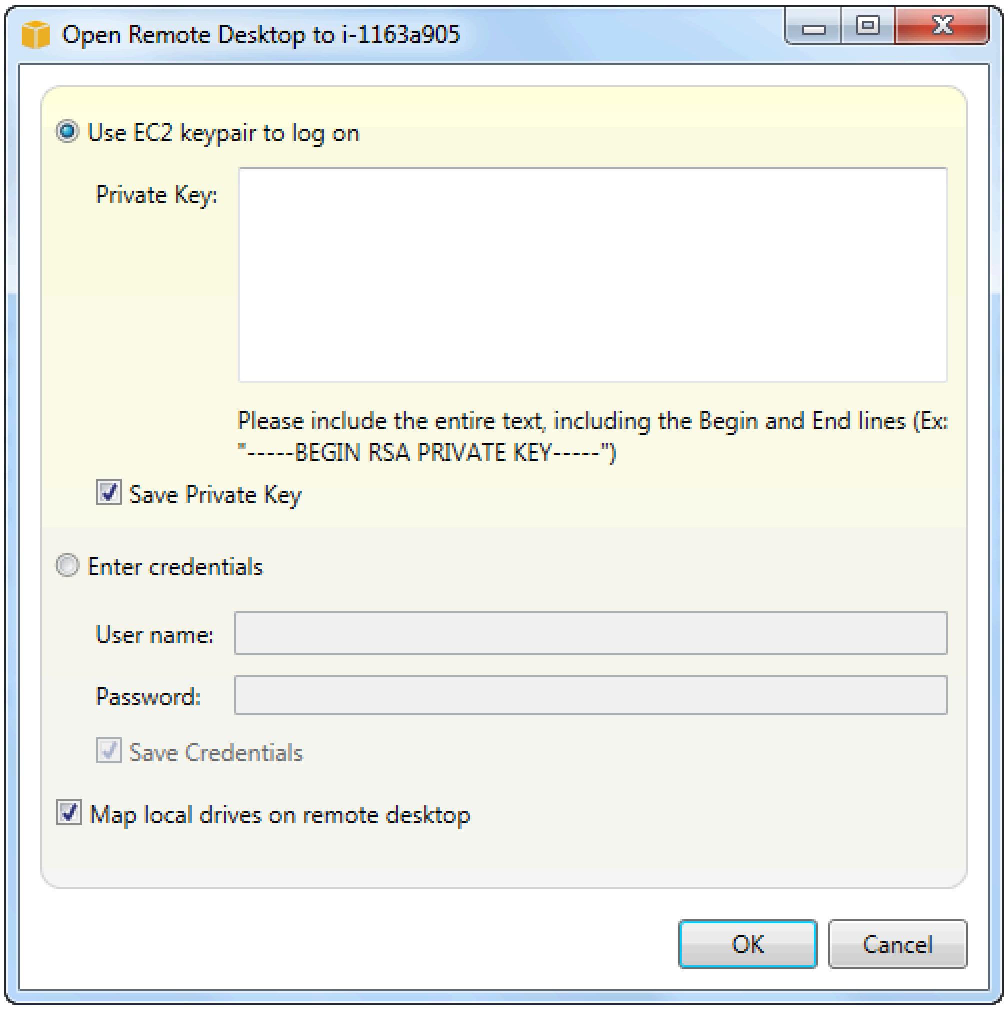
3. Fill in the requested information.
4. Your account information now appears on the **AWS Explorer** tab. When you publish to Elastic Beanstalk, you can select which account you would like to use.

Listing and connecting to server instances

You can view a list of Amazon EC2 instances running your Elastic Beanstalk application environment through the AWS Toolkit for Visual Studio or from the AWS Management Console. You can connect to these instances using Remote Desktop Connection. For information about listing and connecting to your server instances using the AWS Management Console, see Listing and connecting to server instances. The following section steps you through viewing and connecting you to your server instances using the AWS Toolkit for Visual Studio.

To view and connect to Amazon EC2 instances for an environment

1. In Visual Studio, in **AWS Explorer**, expand the **Amazon EC2** node and double-click **Instances**.
2. Right-click the instance ID for the Amazon EC2 instance running in your application's load balancer in the **Instance** column and select **Open Remote Desktop** from the context menu.


Open Remote Desktop to i-1163a905

Use EC2 keypair to log on

Private Key:

Please include the entire text, including the Begin and End lines (Ex: "-----BEGIN RSA PRIVATE KEY-----")

Save Private Key

Enter credentials

User name:

Password:

Save Credentials

Map local drives on remote desktop

3. Select **Use EC2 keypair to log on** and paste the contents of your private key file that you used to deploy your application in the **Private key** box. Alternatively, enter your user name and password in the **User name** and **Password** text boxes.

> **Note**
>
> If the key pair is stored inside the Toolkit, the text box does not appear.

4. Click **OK**.

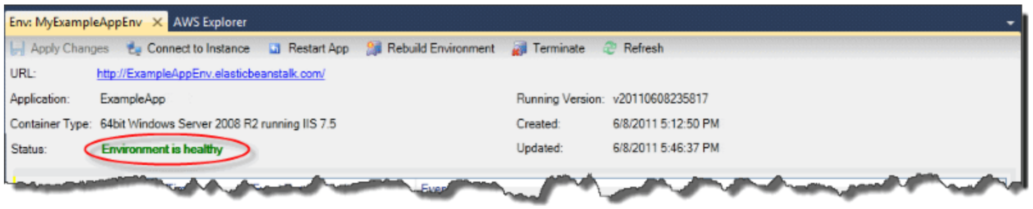
Monitoring application health

When you are running a production website, it is important to know that your application is available and responding to requests. To assist with monitoring your application's responsiveness, Elastic Beanstalk provides features where you can monitor statistics about your application and create alerts that trigger when thresholds are exceeded.

For information about the health monitoring provided by Elastic Beanstalk, see Basic health reporting.

You can access operational information about your application by using either the AWS Toolkit for Visual Studio or the AWS Management Console.

The toolkit displays your environment's status and application health in the **Status** field.

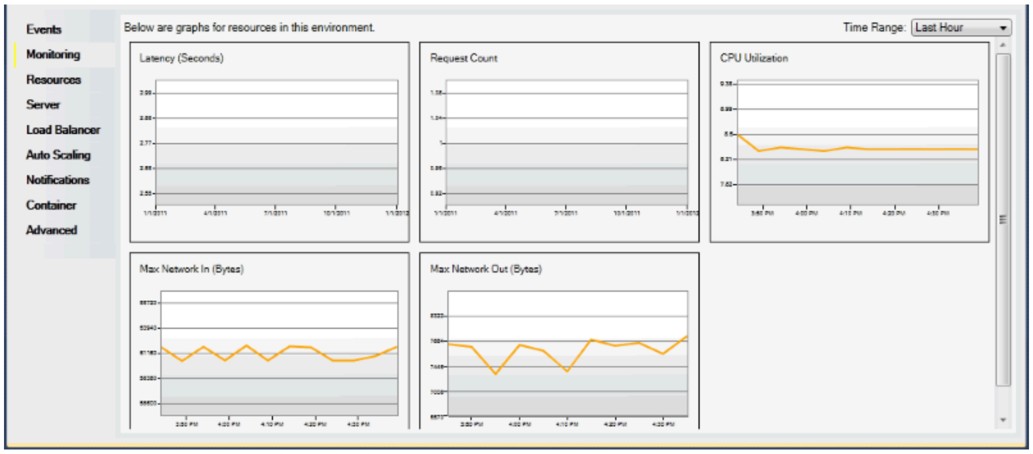


URL: http://ExampleAppEnv.elasticbeanstalk.com/
Application: ExampleApp
Container Type: 64bit Windows Server 2008 R2 running IIS 7.5
Status: Environment is healthy
Running Version: v20110608235817
Created: 6/8/2011 5:12:50 PM
Updated: 6/8/2011 5:46:37 PM

To monitor application health

1. In the AWS Toolkit for Visual Studio, in **AWS Explorer**, expand the Elastic Beanstalk node, and then expand your application node.
2. Right-click your Elastic Beanstalk environment, and then click **View Status**.
3. On your application environment tab, click **Monitoring**.

   The **Monitoring** panel includes a set of graphs showing resource usage for your particular application environment.

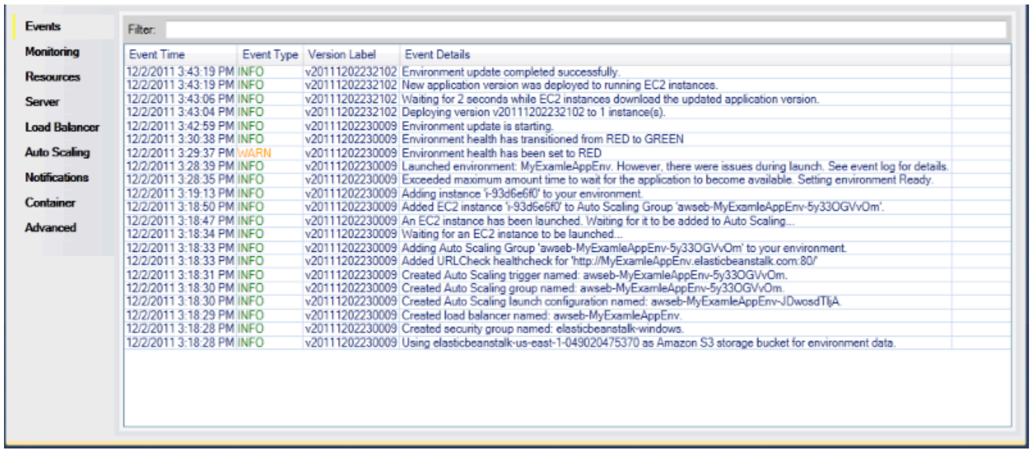


> **Note**
>
> By default, the time range is set to the last hour. To modify this setting, in the **Time Range** list, click a different time range.

You can use the AWS Toolkit for Visual Studio or the AWS Management Console to view events associated with your application.

To view application events

1. In the AWS Toolkit for Visual Studio, in **AWS Explorer**, expand the Elastic Beanstalk node and your application node.
2. Right-click your Elastic Beanstalk environment in **AWS Explorer** and then click **View Status**.
3. In your application environment tab, click **Events**.



Event Time	Event Type	Version Label	Event Details
12/2/2011 3:43:19 PM	INFO	v20111202232102	Environment update completed successfully.
12/2/2011 3:43:19 PM	INFO	v20111202232102	New application version was deployed to running EC2 instances.
12/2/2011 3:43:06 PM	INFO	v20111202232102	Waiting for 2 seconds while EC2 instances download the updated application version.
12/2/2011 3:43:04 PM	INFO	v20111202232102	Deploying version v20111202232102 to 1 instance(s).
12/2/2011 3:42:59 PM	INFO	v20111202230009	Environment update is starting.
12/2/2011 3:30:38 PM	INFO	v20111202230009	Environment health has transitioned from RED to GREEN
12/2/2011 3:29:37 PM	WARN	v20111202230009	Environment health has been set to RED
12/2/2011 3:28:39 PM	INFO	v20111202230009	Launched environment: MyExamleAppEnv. However, there were issues during launch. See event log for details.
12/2/2011 3:28:35 PM	INFO	v20111202230009	Exceeded maximum amount time to wait for the application to become available. Setting environment Ready.
12/2/2011 3:19:13 PM	INFO	v20111202230009	Adding instance 'i-93d6e6f0' to your environment.
12/2/2011 3:18:50 PM	INFO	v20111202230009	Added EC2 instance 'i-93d6e6f0' to Auto Scaling Group 'awseb-MyExamleAppEnv-5y33OGVvOm'.
12/2/2011 3:18:47 PM	INFO	v20111202230009	An EC2 instance has been launched. Waiting for it to be added to Auto Scaling...
12/2/2011 3:18:34 PM	INFO	v20111202230009	Waiting for an EC2 instance to be launched...
12/2/2011 3:18:33 PM	INFO	v20111202230009	Adding Auto Scaling Group 'awseb-MyExamleAppEnv-5y33OGVvOm' to your environment.
12/2/2011 3:18:33 PM	INFO	v20111202230009	Added URLCheck healthcheck for 'http://MyExamleAppEnv.elasticbeanstalk.com:80/'
12/2/2011 3:18:31 PM	INFO	v20111202230009	Created Auto Scaling trigger named: awseb-MyExamleAppEnv-5y33OGVvOm.
12/2/2011 3:18:30 PM	INFO	v20111202230009	Created Auto Scaling group named: awseb-MyExamleAppEnv-5y33OGVvOm.
12/2/2011 3:18:30 PM	INFO	v20111202230009	Created Auto Scaling launch configuration named: awseb-MyExamleAppEnv-JDwosdTIjA.
12/2/2011 3:18:29 PM	INFO	v20111202230009	Created load balancer named: awseb-MyExamleAppEnv.
12/2/2011 3:18:28 PM	INFO	v20111202230009	Created security group named: elasticbeanstalk-windows.
12/2/2011 3:18:28 PM	INFO	v20111202230009	Using elasticbeanstalk-us-east-1-049020475370 as Amazon S3 storage bucket for environment data.

Deploying Elastic Beanstalk applications in .NET using the deployment tool

The AWS Toolkit for Visual Studio includes a deployment tool, a command line tool that provides the same functionality as the deployment wizard in the AWS Toolkit. You can use the deployment tool in your build pipeline or in other scripts to automate deployments to Elastic Beanstalk.

The deployment tool supports both initial deployments and redeployments. If you previously deployed your application using the deployment tool, you can redeploy using the deployment wizard within Visual Studio. Similarly, if you have deployed using the wizard, you can redeploy using the deployment tool.

Note

The deployment tool does not apply recommended values for configuration options like the console or EB CLI. Use configuration files to ensure that any settings that you need are configured when you launch your environment.

This chapter walks you through deploying a sample .NET application to Elastic Beanstalk using the deployment tool, and then redeploying the application using an incremental deployment. For a more in-depth discussion about the deployment tool, including the parameter options, see Deployment Tool.

Prerequisites

To use the deployment tool, you need to install the AWS Toolkit for Visual Studio. For information on prerequisites and installation instructions, see AWS Toolkit for Microsoft Visual Studio.

The deployment tool is typically installed in one of the following directories on Windows:

32-bit	64-bit
`C:\Program Files\AWS Tools\Deployment Tool \awsdeploy.exe`	`C:\Program Files (x86)\AWS Tools\Dep loyment Tool\awsdeploy.exe`

Deploy to Elastic Beanstalk

To deploy the sample application to Elastic Beanstalk using the deployment tool, you first need to modify the `ElasticBeanstalkDeploymentSample.txt` configuration file, which is provided in the `Samples` directory. This configuration file contains the information necessary to deploy your application, including the application name, application version, environment name, and your AWS access credentials. After modifying the configuration file, you then use the command line to deploy the sample application. Your web deploy file is uploaded to Amazon S3 and registered as a new application version with Elastic Beanstalk. It will take a few minutes to deploy your application. Once the environment is healthy, the deployment tool outputs a URL for the running application.

To deploy a .NET application to Elastic Beanstalk

1. From the `Samples` subdirectory where the deployment tool is installed, open `ElasticBeanstalkDeploymentSample.txt` and enter your AWS access key and AWS secret key as in the following example.

```
### AWS Access Key and Secret Key used to create and deploy the application
 instance
AWSAccessKey = AKIAIOSFODNN7EXAMPLE
AWSSecretKey = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Note

For API access, you need an access key ID and secret access key. Use IAM user access keys instead of AWS account root user access keys. For more information about creating access keys, see Manage access keys for IAM users in the *IAM User Guide*.

2. At the command line prompt, type the following:

```
C:\Program Files (x86)\AWS Tools\Deployment Tool>awsdeploy.exe /w Samples
\ElasticBeanstalkDeploymentSample.txt
```

It takes a few minutes to deploy your application. If the deployment succeeds, you will see the message, `Application deployment completed; environment health is Green`.

Note

If you receive the following error, the CNAME already exists.

```
[Error]: Deployment to AWS Elastic Beanstalk failed with exception: DNS name
 (MyAppEnv.elasticbeanstalk.com) is not available.
```

Because a CNAME must be unique, you need to change `Environment.CNAME` in `ElasticBeanstalkDeploymentSample.txt`.

3. In your web browser, navigate to the URL of your running application. The URL will be in the form <CNAME.elasticbeanstalk.com> (e.g., **`MyAppEnv.elasticbeanstalk.com`**).

Migrating your on-premises .NET application to Elastic Beanstalk

AWS Elastic Beanstalk provides a streamlined migration path for your Windows applications running on Internet Information Services (IIS) through the Elastic Beanstalk Command Line Interface (EB CLI). The **eb migrate** command automatically discovers your IIS sites, applications, and virtual directories, preserves their configurations, and deploys them to the AWS Cloud.

This built-in migration capability offers a simpler approach that reduces the complexity and time typically associated with cloud migrations. The migration process helps maintain application functionality and configuration integrity during the transition to AWS.

For complete, detailed instructions about migrating your IIS applications to AWS Elastic Beanstalk, refer to the *Migrating IIS applications* chapter in this guide.

Recommendations for Windows Server retired components on Elastic Beanstalk

This topic provides recommendations if your applications are currently running on the retired Windows Server 2012 R2 platform branches. It also addresses the deprecated support for the TLS 1.0 and 1.1 protocol versions on our AWS service API endpoints and the impacted platform branches.

Windows Server 2012 R2 platform branches retired

Elastic Beanstalk retired Windows Server 2012 R2 platform branches on December 4, 2023, and made the AMIs associated with those platforms private on April 10, 2024. This action prevents the launching of instances in your Windows Server 2012 environments that use the default Beanstalk AMI.

If you have any environments running on retired Windows platform branches we recommend that you migrate them to one of the following Windows Server platforms, which are current and fully supported:

- Windows Server 2022 with IIS 10.0 version 2.x
- Windows Server 2019 with IIS 10.0 version 2.x

For full migration considerations, see Migrating from earlier major versions of the Windows server platform.

For more information about platform deprecation, see Elastic Beanstalk platform support policy.

Note

If you cannot migrate to these fully supported platforms, we recommend using custom AMIs created with Windows Server 2012 R2 or Windows Server 2012 R2 Core AMIs as the base image, if you have not done so already. For detailed instructions, see Preserving access to an Amazon Machine Image (AMI) for a retired platform. Reach out to the AWS Support Center if you need temporary access to an AMI while you perform one of these migration steps.

TLS 1.2 Compatibility

As of December 31, 2023, AWS started fully enforcing TLS 1.2 across all AWS API endpoints. This action removed the ability to use TLS versions 1.0 and 1.1 with all AWS APIs. This information was originally communicated on June 28, 2022. To avoid the risk of availability impact, upgrade any environments running the platform versions identified here to a newer version as soon as possible, if you have not done so already.

Potential impact

Elastic Beanstalk platforms versions that run TLS v1.1 or earlier are impacted. This change impacts environment actions that include but are not limited to the following: configuration deployments, application deployments, auto scaling, new environment launch, log rotation, enhanced health reports, and publishing application logs to the Amazon S3 bucket that's associated with your applications.

Affected Windows Platform Versions

Customers with Elastic Beanstalk environments on the following platform version are advised to upgrade each of their corresponding environments to Windows platform version 2.8.3 or later, released on Feb 18, 2022.

- Windows Server 2019 — platform version 2.8.2 or prior versions

Customers with Elastic Beanstalk environments on the following platform versions are advised to upgrade each of their corresponding environments to Windows platform version 2.10.7 or later, released on Dec 28, 2022.

- Windows Server 2016 — platform version 2.10.6 or prior versions
- Windows Server 2012 — all platform versions; this platform was retired on December 4, 2023
- Windows Server 2008 — all platform versions; this platform was retired on October 28, 2019

For a list of the most recent and supported Windows Server platform versions, see Supported Platforms in the *AWS Elastic Beanstalk Platforms* guide.

For details and best practices about updating your environment, see Updating your Elastic Beanstalk environment's platform version.

Deploying .NET core (Linux) applications with Elastic Beanstalk

Check out the *.NET on AWS Developer Center*

Have you stopped by our *.Net Developer Center*? It's our one stop shop for all things .NET on AWS.
For more information see the .NET on AWS Developer Center.

This chapter provides instructions for configuring and deploying your .NET core (Linux) web application to AWS Elastic Beanstalk. Elastic Beanstalk makes it easy to deploy, manage, and scale your .NET core (Linux) web applications using Amazon Web Services.

You can deploy your application in just a few minutes using the Elastic Beanstalk Command Line Interface (EB CLI) or by using the Elastic Beanstalk console. After you deploy your Elastic Beanstalk application, you can continue to use the EB CLI to manage your application and environment, or you can use the Elastic Beanstalk console, AWS CLI, or the APIs.

Follow the steps in the QuickStart for .NET Core on Linux for step-by-step instructions to create and deploy an ASP.NET Core *Hello World* web application with the EB CLI.

Topics

QuickStart: Deploy a .NET Core on Linux application to Elastic Beanstalk

This QuickStart tutorial walks you through the process of creating a .NET Core on Linux application and deploying it to an AWS Elastic Beanstalk environment.

Note

Tutorial examples are intended for demonstration. Do not use the application for production traffic.

Sections

Your AWS account

If you're not already an AWS customer, you need to create an AWS account. Signing up enables you to access Elastic Beanstalk and other AWS services that you need.

If you already have an AWS account, you can move on to Prerequisites.

Create an AWS account

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Prerequisites

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

EB CLI

This tutorial uses the Elastic Beanstalk Command Line Interface (EB CLI). For details on installing and configuring the EB CLI, see Install EB CLI with setup script (recommended) and Configure the EB CLI.

.NET Core on Linux

If you don't have the .NET SDK installed on your local machine, you can install it by following the Download .NET link on the .NET documentation website.

Verify your .NET SDK installation by running the following command.

```
~$ dotnet --info
```

Step 1: Create a .NET Core on Linux application

Create a project directory.

```
~$ mkdir eb-dotnetcore
~$ cd eb-dotnetcore
```

Next, create a sample Hello World application by running the following commands.

```
~/eb-dotnetcore$ dotnet new web --name HelloElasticBeanstalk
~/eb-dotnetcore$ cd HelloElasticBeanstalk
```

Step 2: Run your application locally

Run the following command to run your application locally.

```
~/eb-dotnetcore/HelloElasticBeasntalk$ dotnet run
```

The output should look something like the following text.

```
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7294
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5052
info: Microsoft.Hosting.Lifetime[0]
```

```
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
```

Note

The `dotnet` command selects a port at random when running the application locally. In this example the port is 5052. When you deploy the application to your Elastic Beanstalk environment, the application will run on port 5000.

Enter the URL address `http://localhost:port` in your web browser. For this specific example, the command is `http://localhost:5052`. The web browser should display “Hello World!”.

Step 3: Deploy your .NET Core on Linux application with the EB CLI

Run the following commands to create an Elastic Beanstalk environment for this application.

To create an environment and deploy your .NET Core on Linux application

1. Compile and publish your application to a folder for deployment to the Elastic Beanstalk environment you're about to create.

```
~$ cd eb-dotnetcore/HelloElasticBeanstalk
~/eb-dotnetcore/HelloElasticBeanstalk$ dotnet publish -o site
```

2. Navigate to the site directory where you just published your app.

```
~/eb-dotnetcore/HelloElasticBeanstalk$ cd site
```

3. Initialize your EB CLI repository with the **eb init** command.

   Be aware of the following details regarding the platform branch version that you specify in the command:

   - Replace `x.y.z` in the following command with the latest version of the platform branch *.NET 6 on AL2023*.
   - To locate the latest platform branch version see .NET Core on Linux *Supported platforms* in the *AWS Elastic Beanstalk Platforms* guide.

- An example of a solution stack name that includes the version number is `64bit-amazon-linux-2023-v3.1.1-running-.net-6`. In this example the branch version is *3.1.1*.

```
~eb-dotnetcore/HelloElasticBeanstalk/site$ eb init -p 64bit-amazon-linux-2023-
vx.y.z-running-.net-6 dotnetcore-tutorial --region us-east-2
Application dotnetcore-tutorial has been created.
```

This command creates an application named `dotnetcore-tutorial` and configures your local repository to create environments with the .NET Core on Linux platform version specified in the command.

4. (Optional) Run **eb init** again to configure a default key pair so that you can use SSH to connect to the EC2 instance running your application.

```
~eb-dotnetcore/HelloElasticBeanstalk/site$ eb init
Do you want to set up SSH for your instances?
(y/n): y
Select a keypair.
1) my-keypair
2) [ Create new KeyPair ]
```

Select a key pair if you have one already, or follow the prompts to create one. If you don't see the prompt or need to change your settings later, run **eb init -i**.

5. Create an environment and deploy your application to it with **eb create**. Elastic Beanstalk automatically builds a zip file for your application and starts it on port 5000.

to

```
~eb-dotnetcore/HelloElasticBeanstalk/site$ eb create dotnet-tutorial
```

It takes about five minutes for Elastic Beanstalk to create your environment.

Step 4: Run your application on Elastic Beanstalk

When the process to create your environment completes, open your website with **eb open**.

```
~eb-dotnetcore/HelloElasticBeanstalk/site$ eb open
```

Congratulations! You've deployed a .NET Core on Linux application with Elastic Beanstalk! This opens a browser window using the domain name created for your application.

Step 5: Clean up

You can terminate your environment when you finish working with your application. Elastic Beanstalk terminates all AWS resources associated with your environment.

To terminate your Elastic Beanstalk environment with the EB CLI run the following command.

```
~eb-dotnetcore/HelloElasticBeanstalk/site$ eb terminate
```

AWS resources for your application

You just created a single instance application. It serves as a straightforward sample application with a single EC2 instance, so it doesn't require load balancing or auto scaling. For single instance applications Elastic Beanstalk creates the following AWS resources:

- **EC2 instance** – An Amazon EC2 virtual machine configured to run web apps on the platform you choose.

  Each platform runs a different set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination thereof. Most platforms use either Apache or nginx as a reverse proxy that processes web traffic in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group** – An Amazon EC2 security group configured to allow incoming traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic is not allowed on other ports.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name** – A domain name that routes to your web app in the form *subdomain*.*region*.*elasticbeanstalk.com*.

Elastic Beanstalk manages all of these resources. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Next steps

After you have an environment running an application, you can deploy a new version of the application or a different application at any time. Deploying a new application version is very quick because it doesn't require provisioning or restarting EC2 instances. You can also explore your new environment using the Elastic Beanstalk console. For detailed steps, see Explore your environment in the *Getting started* chapter of this guide.

After you deploy a sample application or two and are ready to start developing and running .NET Core on Linux applications locally, see Setting up your .NET core on Linux development environment for Elastic Beanstalk.

Deploy with the Elastic Beanstalk console

You can also use the Elastic Beanstalk console to launch the sample application. For detailed steps, see Create an example application in the *Getting started* chapter of this guide.

Setting up your .NET core on Linux development environment for Elastic Beanstalk

This topic provides instructions to set up a .NET core on Linux development environment to test your application locally prior to deploying it to AWS Elastic Beanstalk. It also references websites that provide installation instructions for useful tools.

Sections

Installing the .NET Core SDK

You can use the .NET Core SDK to develop applications that run on Linux.

See the .NET downloads page to download and install the .NET Core SDK.

Installing an IDE

Integrated development environments (IDEs) provide a range of features that facilitate application development. If you haven't used an IDE for .NET development, try Visual Studio Community to get started.

See the Visual Studio Community page to download and install Visual Studio Community.

Installing the AWS Toolkit for Visual Studio

The AWS Toolkit for Visual Studio is an open source plugin for the Visual Studio IDE that makes it easier for developers to develop, debug, and deploy .NET applications using AWS. See the Toolkit for Visual Studio homepage for installation instructions.

Using the Elastic Beanstalk .NET core on Linux platform

This topic describes how to configure, build, and run your .NET core on Linux applications on Elastic Beanstalk.

AWS Elastic Beanstalk supports a number of platform branches for different .NET Core framework versions that run on the Linux operating system. See .NET core on Linux in the *AWS Elastic Beanstalk Platforms* for a full list.

For details about the various ways you can extend an Elastic Beanstalk Linux-based platform, see the section called "Extending Linux platforms".

.NET Core on Linux platform considerations

Proxy server

The Elastic Beanstalk .NET Core on Linux platform includes a reverse proxy that forwards requests to your application. By default, Elastic Beanstalk uses NGINX as the proxy server. You can choose to use no proxy server, and configure Kestrel as your web server. Kestrel is included by default in ASP.NET Core project templates.

Application structure

You can publish *runtime-dependent* applications that use the .NET Core runtime provided by Elastic Beanstalk. You can also publish *self-contained* applications that include the .NET Core runtime and your application's dependencies in the source bundle. To learn more, see the section called "Bundling applications".

Platform configuration

To configure the processes that run on the server instances in your environment, include an optional Procfile in your source bundle. A `Procfile` is required if you have more than one application in your source bundle.

We recommend that you always provide a `Procfile` in the source bundle with your application. This way you precisely control which processes Elastic Beanstalk runs for your application.

Configuration options are available in the Elastic Beanstalk console for modifying the configuration of a running environment. To avoid losing your environment's configuration when you terminate it, you can use saved configurations to save your settings and later apply them to another environment.

To save settings in your source code, you can include configuration files. Settings in configuration files are applied every time you create an environment or deploy your application. You can also use configuration files to install packages, run scripts, and perform other instance customization operations during deployments.

Settings applied in the Elastic Beanstalk console override the same settings in configuration files, if they exist. This lets you have default settings in configuration files, and override them with environment-specific settings in the console. For more information about precedence, and other methods of changing settings, see Configuration options.

Configuring your .NET Core on Linux environment

The .NET Core on Linux platform settings enable you to fine-tune the behavior of your Amazon EC2 instances. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration using the Elastic Beanstalk console.

Use the Elastic Beanstalk console to enable log rotation to Amazon S3 and configure variables that your application can read from the environment.

To configure your .NET Core on Linux environment using the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.

Log options

The **Log Options** section has two settings:

- **Instance profile** – Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3** – Specifies whether log files for your application's Amazon EC2 instances are copied to the Amazon S3 bucket associated with your application.

Environment properties

The **Environment Properties** section enables you to specify environment configuration settings on the Amazon EC2 instances that are running your application. Environment properties are passed in as key-value pairs to the application.

Inside the .NET Core on Linux environment running in Elastic Beanstalk, environment variables are accessible using `Environment.GetEnvironmentVariable("variable-name")`. For example, you could read a property named `API_ENDPOINT` to a variable with the following code.

```
string endpoint = Environment.GetEnvironmentVariable("API_ENDPOINT");
```

See Environment variables and other software settings for more information.

.NET Core on Linux configuration namespace

You can use a configuration file to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be platform specific or apply to all platforms in the Elastic Beanstalk service as a whole. Configuration options are organized into *namespaces*.

The .NET Core on Linux platform supports options in the following namespace, in addition to the options supported for all Elastic Beanstalk environments:

- `aws:elasticbeanstalk:environment:proxy` – Choose to use NGINX or no proxy server. Valid values are `nginx` or `none`.

The following example configuration file shows the use of the .NET Core on Linux-specific configuration options.

Example .ebextensions/proxy-settings.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: none
```

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See Configuration options for more information.

Bundling applications for the .NET Core on Linux Elastic Beanstalk platform

You can run both *runtime-dependent* and *self-contained* .NET Core applications on AWS Elastic Beanstalk.

A runtime-dependent application uses a .NET Core runtime that Elastic Beanstalk provides to run your application. Elastic Beanstalk uses the `runtimeconfig.json` file in your source bundle to determine the runtime to use for your application. Elastic Beanstalk chooses the latest compatible runtime available for your application.

A self-contained application includes the .NET Core runtime, your application, and its dependencies. To use a version of the .NET Core runtime that Elastic Beanstalk doesn't include in its platforms, provide a self-contained application.

Examples

You can compile both self-contained and runtime-dependent applications with the `dotnet publish` command. To learn more about publishing .NET Core apps, see .NET Core application publishing overview in the .NET Core documentation.

The following example file structure defines a single application that uses a .NET Core runtime that Elastic Beanstalk provides.

```
### appsettings.Development.json
### appsettings.json
### dotnetcoreapp.deps.json
### dotnetcoreapp.dll
### dotnetcoreapp.pdb
### dotnetcoreapp.runtimeconfig.json
### web.config
### Procfile
### .ebextensions
### .platform
```

You can include multiple applications in your source bundle. The following example defines two applications to run on the same web server. To run multiple applications, you must include a Procfile in your source bundle. For a full example application, see dotnet-core-linux-multiple-apps.zip.

```
### DotnetMultipleApp1
#   ### Amazon.Extensions.Configuration.SystemsManager.dll
#   ### appsettings.Development.json
#   ### appsettings.json
#   ### AWSSDK.Core.dll
#   ### AWSSDK.Extensions.NETCore.Setup.dll
#   ### AWSSDK.SimpleSystemsManagement.dll
#   ### DotnetMultipleApp1.deps.json
#   ### DotnetMultipleApp1.dll
#   ### DotnetMultipleApp1.pdb
#   ### DotnetMultipleApp1.runtimeconfig.json
#   ### Microsoft.Extensions.PlatformAbstractions.dll
#   ### Newtonsoft.Json.dll
#   ### web.config
### DotnetMultipleApp2
#   ### Amazon.Extensions.Configuration.SystemsManager.dll
#   ### appsettings.Development.json
#   ### appsettings.json
#   ### AWSSDK.Core.dll
#   ### AWSSDK.Extensions.NETCore.Setup.dll
#   ### AWSSDK.SimpleSystemsManagement.dll
#   ### DotnetMultipleApp2.deps.json
#   ### DotnetMultipleApp2.dll
#   ### DotnetMultipleApp2.pdb
#   ### DotnetMultipleApp2.runtimeconfig.json
#   ### Microsoft.Extensions.PlatformAbstractions.dll
#   ### Newtonsoft.Json.dll
#   ### web.config
### Procfile
### .ebextensions
### .platform
```

Using a Procfile to configure your .NET Core on Linux Elastic Beanstalk environment

To run multiple applications on the same web server, you must include a `Procfile` in your source bundle that tells Elastic Beanstalk which applications to run.

We recommend that you always provide a `Procfile` in the source bundle with your application. This way you precisely control which processes Elastic Beanstalk runs for your application and which arguments these processes receive.

The following example uses a `Procfile` to specify two applications for Elastic Beanstalk to run on the same web server.

Example Procfile

```
web: dotnet ./dotnet-core-app1/dotnetcoreapp1.dll
web2: dotnet ./dotnet-core-app2/dotnetcoreapp2.dll
```

For details about writing and using a `Procfile`, see Buildfile and Procfile.

Configuring the proxy server

AWS Elastic Beanstalk uses NGINX as the reverse proxy to relay requests to your application. Elastic Beanstalk provides a default NGINX configuration that you can either extend or override completely with your own configuration.

By default, Elastic Beanstalk configures the NGINX proxy to forward requests to your application on port 5000. You can override the default port by setting the `PORT` environment property to the port on which your main application listens.

Note

The port that your application listens on doesn't affect the port that the NGINX server listens on to receive requests from the load balancer.

Configuring the proxy server on your platform version

All AL2023/AL2 platforms support a uniform proxy configuration feature. For more information about configuring the proxy server on your platform versions running AL2023/AL2, see Reverse proxy configuration.

The following example configuration file extends your environment's NGINX configuration. The configuration directs requests to `/api` to a second web application that listens on port 5200 on the web server. By default, Elastic Beanstalk forwards requests to a single application that listens on port 5000.

Example `01_custom.conf`

```
location /api {
     proxy_pass          http://127.0.0.1:5200;
     proxy_http_version  1.1;

     proxy_set_header   Upgrade $http_upgrade;
     proxy_set_header   Connection $http_connection;
     proxy_set_header   Host $host;
     proxy_cache_bypass $http_upgrade;
     proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
     proxy_set_header   X-Forwarded-Proto $scheme;
}
```

The AWS Toolkit for Visual Studio - Working with .Net Core on Elastic Beanstalk

This topic shows how you can do the following tasks using the AWS Toolkit for Visual Studio:

- Create an ASP.NET Core web application using a Visual Studio template.
- Create an Elastic Beanstalk Amazon Linux environment.
- Deploy the ASP.NET Core web application to the new Amazon Linux environment.

This topic also explores how you can use the AWS Toolkit for Visual Studio to manage your Elastic Beanstalk application environments and monitor your application's health.

The AWS Toolkit for Visual Studio is a plugin to the Visual Studio IDE. With the toolkit you can deploy and manage applications in Elastic Beanstalk while you are working in your Visual Studio environment.

Sections

Prerequisites

Before you begin this tutorial, you need to install the AWS Toolkit for Visual Studio. For instructions, see Setting up the AWS Toolkit for Visual Studio.

If you have never used the toolkit before, the first thing you'll need to do after installing the toolkit is to register your AWS credentials with the toolkit. For more information about this, see Providing AWS Credentials.

Create a new application project

If you don't have a .NET Core application project in Visual Studio, you can easily create one using one of the Visual Studio project templates.

To create a new ASP.NET Core web application project

1. In Visual Studio, on the **File** menu, choose **New** and then choose **Project**.
2. In the **Create a new project** dialog box, select **C#**, select **Linux**, and then select **Cloud**.
3. From the list of project templates that displays select **ASP.NET Core Web Application**, and then select **Next**.

   Note

   If you don't see **ASP.NET Core Web Application** listed in the project templates, you can install it in Visual Studio.

   1. Scroll to the bottom of the template list and select the **Install more tools and features** link that is located under the template list.

2. If you are prompted to allow the Visual Studio application to make changes to your device, select **Yes**.
3. Choose the **Workloads** tab, then select **ASP.NET and web development.**
4. Select the **Modify** button. The **Visual Studio Installer** installs the project template.
5. After the installer completes, exit the panel to return to where you left off in Visual Studio .

4. In the **Configure your new project** dialog box, enter a **Project name**. The **Solution name** defaults to your project name. Next, choose **Create**.
5. In the **Create a new ASP.NET Core web application** dialog box, select **.NET Core**, and then select **ASP.NET Core 3.1**. From the list of application types that are displayed, select **Web Application**, then select the **Create** button.

Visual Studio displays the **Creating Project** dialog box when it creates your application. After Visual Studio completes generating your application, a panel with your application name is displayed.

Create an Elastic Beanstalk environment and deploy your application

This section describes how to create an Elastic Beanstalk environment for your application and deploy your application to that environment.

To create a new environment and deploy your application

1. In Visual Studio select **View**, then **Solution Explorer**.
2. In **Solution Explorer**, open the context (right-click) menu for your application, and then select **Publish to AWS Elastic Beanstalk**.
3. In the **Publish to AWS Elastic Beanstalk** wizard, enter your account information.

   a. For **Account profile to use**, select your **default** account or choose the **Add another account** icon to enter new account information.

   b. For **Region**, select the Region where you want to deploy your application. For information about available AWS Regions, see AWS Elastic Beanstalk Endpoints and Quotas in the *AWS General Reference*. If you select a Region that is not supported by Elastic Beanstalk, then the option to deploy to Elastic Beanstalk is unavailable.

   c. Select **Create a new application environment**, then choose **Next**.

4. On the **Application Environment** dialog box, enter the details for your new application environment.

5. On the next **AWS** options dialog box, set Amazon EC2 options and other AWS related options for your deployed application.

   a. For **Container type** select **64bit Amazon Linux 2 v*<n.n.n>* running .NET Core.**

      Note

      We recommend you select the current platform version of Linux. This version contains the most recent security and bug fixes that are included in our latest Amazon Machine Image (AMI).

   b. For **Instance Type**, select **t2.micro**. (Choosing a micro instance type minimizes the cost associated with running the instance.)

   c. For **Key pair**, select **Create new key pair**. Enter a name for the new key pair, and then choose **OK**. (In this example, we use **`myuseastkeypair`**.) A key pair enables remote-desktop access to your Amazon EC2 instances. For more information about Amazon EC2 key pairs, see Using Credentials in the *Amazon Elastic Compute Cloud User Guide*.

   d. For a simple, low traffic application, select **Single instance environment**. For more information, see Environment types

   e. Select **Next**.

   For more information about the AWS options that are not used in this example consider the following pages:

   - For **Use custom AMI**, see Using a custom Amazon machine image (AMI) in your Elastic Beanstalk environment.
   - If you don't select **Single instance environment**, you need to choose a **Load balance type**. See Load balancer for your Elastic Beanstalk environment for more information.
   - Elastic Beanstalk uses the default Amazon VPC (Amazon Virtual Private Cloud) configuration if you didn't choose **Use non-default VPC**. For more information see Using Elastic Beanstalk with Amazon VPC.
   - Choosing the **Enable Rolling Deployments** option splits a deployment into batches to avoid potential downtime during deployments. For more information, see Deploying applications to Elastic Beanstalk environments.

- Choosing the **Relational Database Access** option connects your Elastic Beanstalk environment to a previously created Amazon RDS database with *Amazon RDS DB Security Groups*. For more information, see Controlling Access with Security Groups in the *Amazon RDS User Guide*.

6. Select **Next** on the **Permissions** dialog box.
7. Select **Next** on the **Applications Options** dialog box.
8. Review your deployment options. After you've verified your settings are correct, select **Deploy**.

Your ASP.NET Core web application is exported as a web deploy file. This file is then uploaded to Amazon S3 and registered as a new application version with Elastic Beanstalk. The Elastic Beanstalk deployment feature monitors your environment until it is available with the newly deployed code. The **Status** for your environment is displayed on the Env:<environment name> tab. After the status updates to **Environment is healthy**, you can select the URL address to launch the web application.

Terminating an environment

To avoid incurring charges for unused AWS resources, you can use the AWS Toolkit for Visual Studio to terminate a running environment.

Note

You can always launch a new environment using the same version later.

To terminate an environment

1. Expand the Elastic Beanstalk node and the application node. In **AWS Explorer** open the context (right-click) menu for your application environment and select **Terminate Environment**.
2. When prompted, select **Yes** to confirm that you want to terminate the environment. It takes a few minutes for Elastic Beanstalk to terminate the AWS resources running in the environment.

The **Status** for your environment on the Env:<environment name> tab changes to **Terminating** and is eventually **Terminated**.

Note

When you terminate your environment, the CNAME associated with the terminated environment becomes available for anyone to use.

Managing your Elastic Beanstalk application environments

This section describes the specific service settings you can edit in the AWS Toolkit for Visual Studio as part of your application environment configuration. With the AWS Toolkit for Visual Studio and the AWS Management Console, you can change the provisioning and configuration of the AWS resources used by your application environments. For information on how to manage your application environments using the AWS Management Console, see Creating environments in Elastic Beanstalk.

Changing environment configurations settings

When you deploy your application, Elastic Beanstalk configures several connected AWS cloud computing services. You can control how these individual services are configured by using the AWS Toolkit for Visual Studio.

To edit an application's environment settings

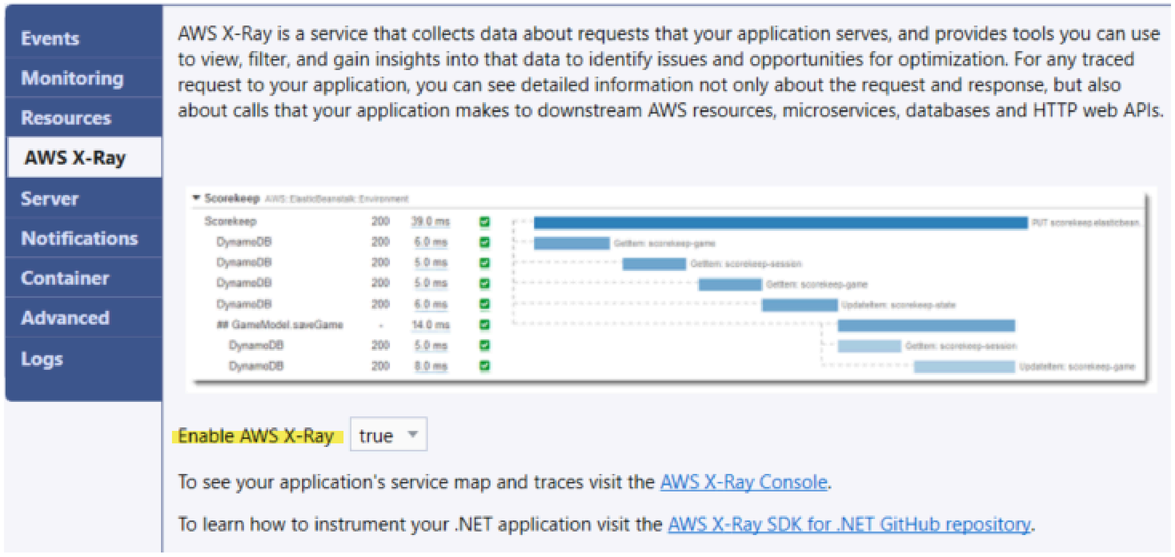
1. In Visual Studio, on the **File** menu, choose **AWS Explorer**.
2. Expand the Elastic Beanstalk node and your application node. Open the context (right-click) menu for your application environment and select **View Status**.

   You can now configure settings for the following:

   - AWS X-Ray
   - Server
   - Load Balancer (only applies to multiple-instance environments)
   - Auto Scaling (only applies to multiple-instance environments)
   - Notifications
   - Container
   - Advanced Configuration Options

Configuring AWS X-Ray using the AWS toolkit for Visual Studio

AWS X-Ray provides request tracing, exception collection, and profiling capabilities. With the AWS X-Ray panel, you can enable or disable X-Ray for your application. For more information about X-Ray, see the AWS X-Ray Developer Guide.



AWS X-Ray is a service that collects data about requests that your application serves, and provides tools you can use to view, filter, and gain insights into that data to identify issues and opportunities for optimization. For any traced request to your application, you can see detailed information not only about the request and response, but also about calls that your application makes to downstream AWS resources, microservices, databases and HTTP web APIs.

```
▼ Scorekeep AWS::ElasticBeanstalk::Environment
Scorekeep              200   39.0 ms   PUT scorekeep.elasticbean...
  DynamoDB             200    6.0 ms   GetItem: scorekeep-game
  DynamoDB             200    5.0 ms   GetItem: scorekeep-session
  DynamoDB             200    5.0 ms   GetItem: scorekeep-game
  DynamoDB             200    6.0 ms   UpdateItem: scorekeep-state
  ## GameModel.saveGame  -   14.0 ms
    DynamoDB           200    5.0 ms   GetItem: scorekeep-session
    DynamoDB           200    8.0 ms   UpdateItem: scorekeep-game
```

Enable AWS X-Ray: true

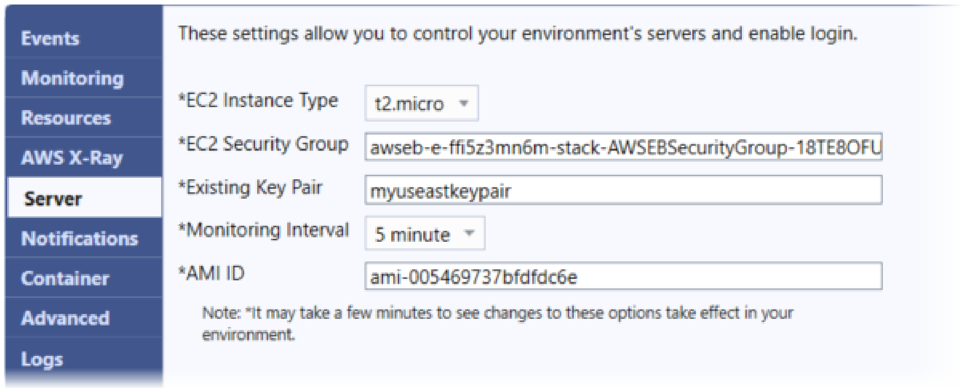
To see your application's service map and traces visit the AWS X-Ray Console.

To learn how to instrument your .NET application visit the AWS X-Ray SDK for .NET GitHub repository.

Configuring EC2 instances using the AWS toolkit for Visual Studio

You can use Amazon Elastic Compute Cloud (Amazon EC2) to launch and manage server instances in Amazon's data centers. You can use Amazon EC2 server instances at any time, for as long as you need, and for any legal purpose. Instances are available in different sizes and configurations. For more information, see Amazon EC2.

You can edit your Amazon EC2 instance configuration with the **Server** tab inside your application environment tab in the AWS Toolkit for Visual Studio.



These settings allow you to control your environment's servers and enable login.

*EC2 Instance Type	t2.micro
*EC2 Security Group	awseb-e-ffi5z3mn6m-stack-AWSEBSecurityGroup-18TE8OFU
*Existing Key Pair	myuseastkeypair
*Monitoring Interval	5 minute
*AMI ID	ami-005469737bfdfdc6e

Note: *It may take a few minutes to see changes to these options take effect in your environment.

Amazon EC2 instance types

Instance type displays the instance types available to your Elastic Beanstalk application. Change the instance type to select a server with the characteristics (including memory size and CPU power) that are most appropriate to your application. For example, applications with intensive and long-running operations can require more CPU or memory.

For more information about the Amazon EC2 instance types available for your Elastic Beanstalk application, see Instance Types in the *Amazon Elastic Compute Cloud User Guide*.

Amazon EC2 security groups

You can control access to your Elastic Beanstalk application using an *Amazon EC2 Security Group*. A security group defines firewall rules for your instances. These rules specify which incoming network traffic should be delivered to your instance. All other incoming traffic is discarded. You can modify rules for a group at any time. The new rules are automatically enforced for all running instances and instances launched in the future.

You can specify which Amazon EC2 Security Groups control access to your Elastic Beanstalk application. To do this, enter the names of specific Amazon EC2 security groups (separating multiple secruity groups with commas) into the **EC2 Security Groups** text box. You can do this either by using the AWS Management Console or the AWS Toolkit for Visual Studio.

To create a security group using the AWS toolkit for Visual Studio

1. In Visual Studio, in **AWS Explorer**, expand the **Amazon EC2** node, and then select **Security Groups**.
2. Select **Create Security Group**, and enter a name and description for your security group.
3. Select **OK**.

For more information on Amazon EC2 Security Groups, see Using Security Groups in the *Amazon Elastic Compute Cloud User Guide*.

Amazon EC2 key pairs

You can securely log in to the Amazon EC2 instances provisioned for your Elastic Beanstalk application with an Amazon EC2 key pair.

Important

You must create an Amazon EC2 key pair and configure your Amazon EC2 instances provisioned by Elastic Beanstalk to be able to access these instances. You can create your key pair using the **Publish to AWS** wizard inside the AWS Toolkit for Visual Studio when you deploy your application to Elastic Beanstalk. If you want to create additional key pairs using the Toolkit, follow the steps described here. Alternatively, you can set up your Amazon EC2 key pairs using the AWS Management Console. For instructions on creating a key pair for Amazon EC2, see the Amazon Elastic Compute Cloud Getting Started Guide.

The **Existing Key Pair** text box lets you specify the name of an Amazon EC2 key pair that you can use to securely log in to the Amazon EC2 instances that are running your Elastic Beanstalk application.

To specify the name of an Amazon EC2 key pair

1. Expand the **Amazon EC2** node and select **Key Pairs**.
2. Select **Create Key Pair** and enter the key pair name.
3. Select **OK**.

For more information about Amazon EC2 key pairs, go to Using Amazon EC2 Credentials in the *Amazon Elastic Compute Cloud User Guide*. For more information about connecting to Amazon EC2 instances, see

Monitoring interval

By default, only basic Amazon CloudWatch metrics are enabled. They return data in five-minute periods. You can enable more granular one-minute CloudWatch metrics by selecting **1 minute** for the **Monitoring Interval** in the **Server** section of the **Configuration** tab for your environment in the AWS Toolkit for Eclipse.

Note

Amazon CloudWatch service charges can apply for one-minute interval metrics. See Amazon CloudWatch for more information.

Custom AMI ID

You can override the default AMI used for your Amazon EC2 instances with your own custom AMI by entering the identifier of your custom AMI into the **Custom AMI ID** box in the **Server** section of the **Configuration** tab for your environment in the AWS Toolkit for Eclipse.

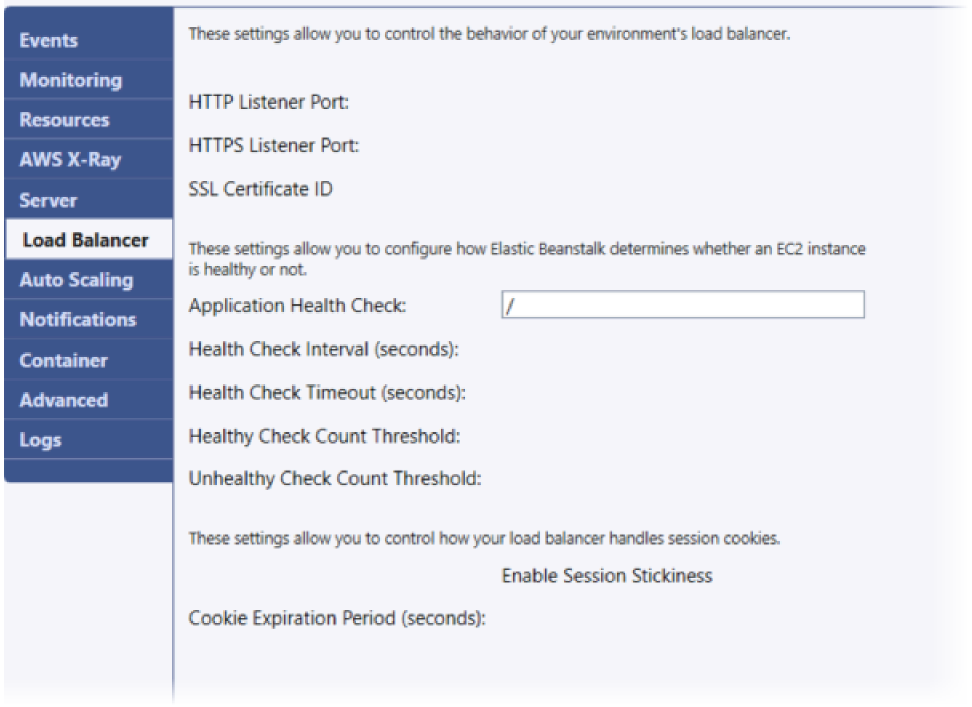
> **Important**
>
> Using your own AMI is an advanced task that you should do with care. If you need a custom AMI, we recommend you start with the default Elastic Beanstalk AMI and then modify it. To be considered healthy, Elastic Beanstalk expects Amazon EC2 instances to meet a set of requirements, including having a running host manager. If these requirements are not met, your environment might not work properly.

Configuring Elastic Load Balancing using the AWS toolkit for Visual Studio

Elastic Load Balancing is an Amazon web service that helps you improve the availability and scalability of your application. This service makes it easy for you to distribute application loads between two or more Amazon EC2 instances. Elastic Load Balancing improves availability through providing additional redundancy and supports traffic growth for your application.

With Elastic Load Balancing, you can automatically distribute and balance incoming application traffic among all your running instances. You can also easily add new instances when increasing the capacity of your application is required.

Elastic Beanstalk automatically provisions Elastic Load Balancing when you deploy an application. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration with the **Load Balancer** tab inside your application environment tab in AWS Toolkit for Visual Studio.



These settings allow you to control the behavior of your environment's load balancer.

HTTP Listener Port:

HTTPS Listener Port:

SSL Certificate ID

These settings allow you to configure how Elastic Beanstalk determines whether an EC2 instance is healthy or not.

Application Health Check: /

Health Check Interval (seconds):

Health Check Timeout (seconds):

Healthy Check Count Threshold:

Unhealthy Check Count Threshold:

These settings allow you to control how your load balancer handles session cookies.

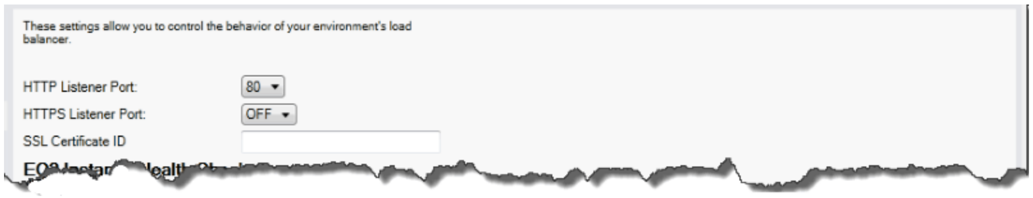
Enable Session Stickiness

Cookie Expiration Period (seconds):

The following sections describe the Elastic Load Balancing parameters you can configure for your application.

Ports

The load balancer provisioned to handle requests for your Elastic Beanstalk application sends requests to the Amazon EC2 instances that are running your application. The provisioned load balancer can listen for requests on HTTP and HTTPS ports and route requests to the Amazon EC2 instances in your AWS Elastic Beanstalk application. By default, the load balancer handles requests on the HTTP port. For this to work, at least one of the ports (either HTTP or HTTPS) must be turned on.



These settings allow you to control the behavior of your environment's load balancer.

HTTP Listener Port: 80

HTTPS Listener Port: OFF

SSL Certificate ID

> **Important**
>
> Make sure that the port that you specified is not locked down; otherwise, you won't be able to connect to your Elastic Beanstalk application.

Controlling the HTTP port

To turn off the HTTP port, select **OFF** for **HTTP Listener Port**. To turn on the HTTP port, you select an HTTP port (for example, **80**) from the list.

Note

To access your environment using a port other than the default port 80, such as port 8080, add a listener to the existing load balancer and configure the new listener to listen on that port.
For example, using the AWS CLI for Classic load balancers, type the following command, replacing *LOAD_BALANCER_NAME* with the name of your load balancer for Elastic Beanstalk.

```
aws elb create-load-balancer-listeners --load-balancer-name LOAD_BALANCER_NAME
 --listeners "Protocol=HTTP, LoadBalancerPort=8080, InstanceProtocol=HTTP,
 InstancePort=80"
```

For example, using the AWS CLI for Application Load Balancers, type the following command, replacing *LOAD_BALANCER_ARN* with the ARN of your load balancer for Elastic Beanstalk.

```
aws elbv2 create-listener --load-balancer-arn LOAD_BALANCER_ARN --protocol HTTP
 --port 8080
```

If you want Elastic Beanstalk to monitor your environment, do not remove the listener on port 80.

Controlling the HTTPS port

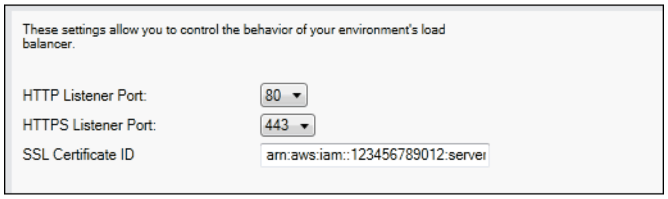
Elastic Load Balancing supports the HTTPS/TLS protocol to enable traffic encryption for client connections to the load balancer. Connections from the load balancer to the EC2 instances use plaintext encryption. By default, the HTTPS port is turned off.

To turn on the HTTPS port

1. Create a new certificate using AWS Certificate Manager (ACM) or upload a certificate and key to AWS Identity and Access Management (IAM). For more information about requesting an

ACM certificate, see Request a Certificate in the *AWS Certificate Manager User Guide*. For more information about importing third-party certificates into ACM, see Importing Certificates in the *AWS Certificate Manager User Guide*. If ACM is not available in your region, use AWS Identity and Access Management (IAM) to upload a third-party certificate. The ACM and IAM services store the certificate and provide an Amazon Resource Name (ARN) for the SSL certificate. For more information about creating and uploading certificates to IAM, see Working with Server Certificates in *IAM User Guide*.

2. Specify the HTTPS port by selecting a port for **HTTPS Listener Port**.



These settings allow you to control the behavior of your environment's load balancer.

HTTP Listener Port:	80
HTTPS Listener Port:	443
SSL Certificate ID	arn:aws:iam::123456789012:server

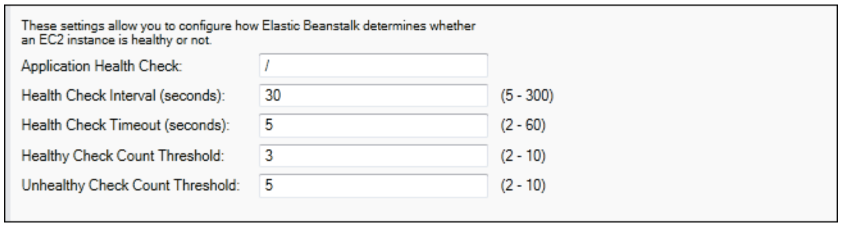
3. For **SSL Certificate ID**, enter the Amazon Resources Name (ARN) of your SSL certificate. For example, **`arn:aws:iam::123456789012:server-certificate/abc/certs/build`** or **`arn:aws:acm:us-east-2:123456789012:certificate/12345678-12ab-34cd-56ef-12345678`**. Use the SSL certificate that you created or uploaded in step 1.

To turn off the HTTPS port, select **OFF** for **HTTPS Listener Port**.

Health checks

The health check definition includes a URL to be queried for instance health. By default, Elastic Beanstalk uses TCP:80 for nonlegacy containers and HTTP:80 for legacy containers. You can override the default URL to match an existing resource in your application (for example, `/myapp/default.aspx`) by entering it in the **Application Health Check URL** box. If you override the default URL, then Elastic Beanstalk uses HTTP to query the resource. To check if you are using a legacy container type, see the section called “Why are some platform versions marked legacy?”

You can control the settings for the health check using the **EC2 Instance Health Check** section of the **Load Balancing** panel.



These settings allow you to configure how Elastic Beanstalk determines whether an EC2 instance is healthy or not.

Application Health Check:	/	
Health Check Interval (seconds):	30	(5 - 300)
Health Check Timeout (seconds):	5	(2 - 60)
Healthy Check Count Threshold:	3	(2 - 10)
Unhealthy Check Count Threshold:	5	(2 - 10)

The health check definition includes a URL to be queried for instance health. Override the default URL to match an existing resource in your application (for example, `/myapp/index.jsp`) by entering it in the **Application Health Check URL** box.

The following list describes the health check parameters you can set for your application.

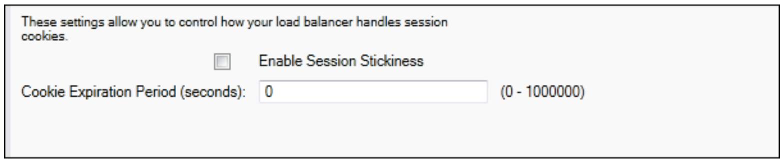
- For **Health Check Interval (seconds)**, enter the number of seconds Elastic Load Balancing waits between health checks for your application's Amazon EC2 instances.
- For **Health Check Timeout (seconds)**, specify the number of seconds Elastic Load Balancing waits for a response before it considers the instance unresponsive.
- For **Healthy Check Count Threshold** and **Unhealthy Check Count Threshold**, specify the number of consecutive successful or unsuccessful URL probes before Elastic Load Balancing changes the instance health status. For example, specifying **5** for **Unhealthy Check Count Threshold** means that the URL must return an error message or timeout five consecutive times before Elastic Load Balancing considers the health check as failed.

Sessions

By default, a load balancer routes each request independently to the server instance with the smallest load. By comparison, a sticky session binds a user's session to a specific server instance so that all requests coming from the user during the session are sent to the same server instance.

Elastic Beanstalk uses load balancer–generated HTTP cookies when sticky sessions are enabled for an application. The load balancer uses a special load balancer–generated cookie to track the application instance for each request. When the load balancer receives a request, it first checks to see if this cookie is present in the request. If it is present, the request is sent to the application instance that is specified in the cookie. If there is no cookie, the load balancer chooses an application instance based on the existing load balancing algorithm. A cookie is inserted into the response for binding subsequent requests from the same user to that application instance. The policy configuration defines a cookie expiry, which establishes the duration of validity for each cookie.

You can use the **Sessions** section on the **Load Balancer** tab to specify whether the load balancer for your application allows session stickiness.


These settings allow you to control how your load balancer handles session cookies.

☐ Enable Session Stickiness

Cookie Expiration Period (seconds): 0 (0 - 1000000)

For more information on Elastic Load Balancing, see the Elastic Load Balancing Developer Guide.

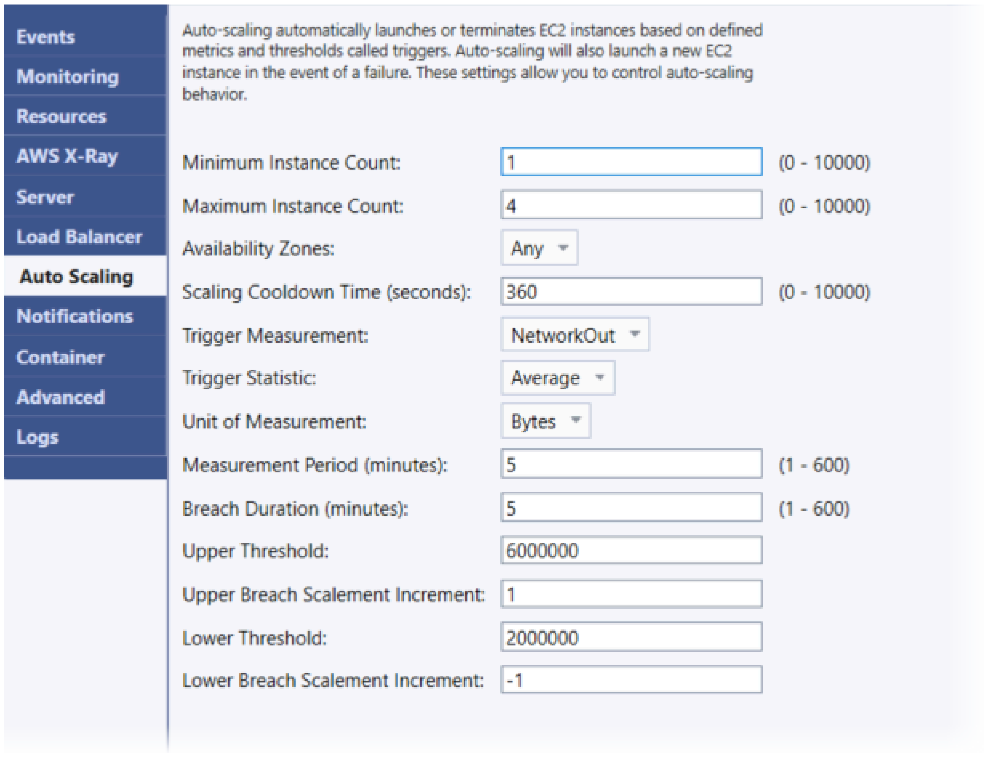
Configuring Auto Scaling using the AWS toolkit for Visual Studio

Amazon EC2 Auto Scaling is an Amazon web service that is designed to automatically launch or terminate Amazon EC2 instances based on user-defined triggers. You can set up *Auto Scaling groups* and associate *triggers* with these groups to automatically scale computing resources based on metrics such as bandwidth usage or CPU utilization. Amazon EC2 Auto Scaling works with Amazon CloudWatch to retrieve metrics for the server instances running your application.

Amazon EC2 Auto Scaling lets you take a group of Amazon EC2 instances and set various parameters to have this group automatically increase or decrease in number. Amazon EC2 Auto Scaling can add or remove Amazon EC2 instances from that group to help you seamlessly deal with traffic changes to your application.

Amazon EC2 Auto Scaling also monitors the health of each Amazon EC2 instance that it launches. If any instance terminates unexpectedly, Amazon EC2 Auto Scaling detects the termination and launches a replacement instance. This capability enables you to maintain a fixed, desired number of Amazon EC2 instances automatically.

Elastic Beanstalk provisions Amazon EC2 Auto Scaling for your application. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration with the **Auto Scaling** tab inside your application environment tab in the AWS Toolkit for Visual Studio.



Auto-scaling automatically launches or terminates EC2 instances based on defined metrics and thresholds called triggers. Auto-scaling will also launch a new EC2 instance in the event of a failure. These settings allow you to control auto-scaling behavior.

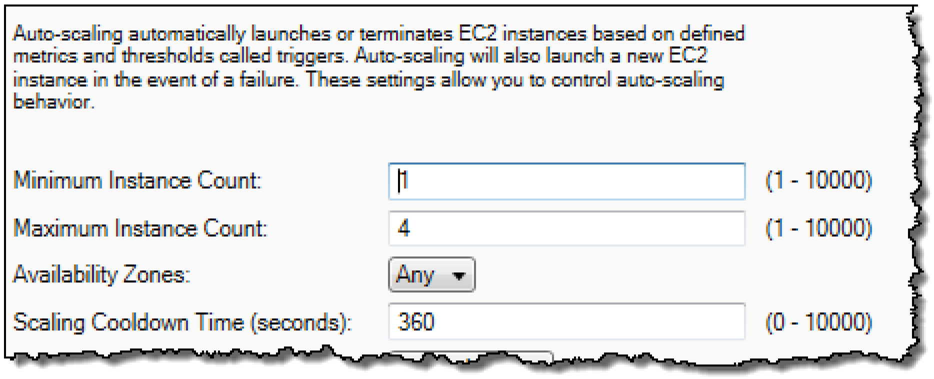
Minimum Instance Count:	1	(0 - 10000)
Maximum Instance Count:	4	(0 - 10000)
Availability Zones:	Any	
Scaling Cooldown Time (seconds):	360	(0 - 10000)
Trigger Measurement:	NetworkOut	
Trigger Statistic:	Average	
Unit of Measurement:	Bytes	
Measurement Period (minutes):	5	(1 - 600)
Breach Duration (minutes):	5	(1 - 600)
Upper Threshold:	6000000	
Upper Breach Scalement Increment:	1	
Lower Threshold:	2000000	
Lower Breach Scalement Increment:	-1	

The following section discusses how to configure Auto Scaling parameters for your application.

Launch the configuration

You can edit the launch configuration to control how your Elastic Beanstalk application provisions Amazon EC2 Auto Scaling resources.

The **Minimum Instance Count** and **Maximum Instance Count** boxes let you specify the minimum and maximum size of the Auto Scaling group that your Elastic Beanstalk application uses.



Auto-scaling automatically launches or terminates EC2 instances based on defined metrics and thresholds called triggers. Auto-scaling will also launch a new EC2 instance in the event of a failure. These settings allow you to control auto-scaling behavior.

Minimum Instance Count:	1	(1 - 10000)
Maximum Instance Count:	4	(1 - 10000)
Availability Zones:	Any	
Scaling Cooldown Time (seconds):	360	(0 - 10000)

> **Note**
>
> To maintain a fixed number of Amazon EC2 instances, set **Minimum Instance Count** and **Maximum Instance Count** to the same value.

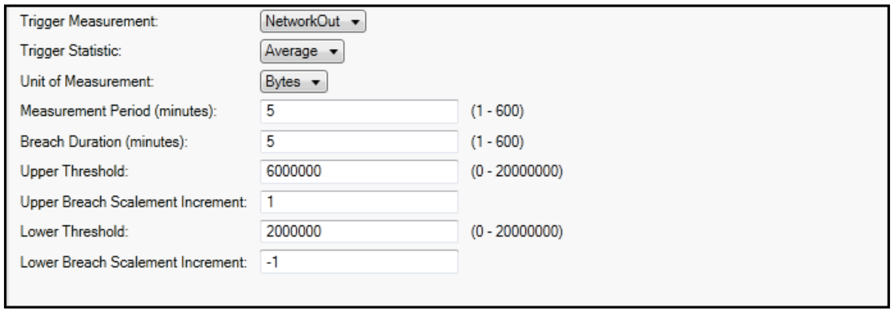
The **Availability Zones** box lets you specify the number of Availability Zones you want your Amazon EC2 instances to be in. It is important to set this number if you want to build fault-tolerant applications. If one Availability Zone goes down, your instances will still run in your other Availability Zones.

> **Note**
>
> Currently, it is not possible to specify which Availability Zone your instance will be in.

Triggers

A *trigger* is an Amazon EC2 Auto Scaling mechanism that you set to tell the system when you want to increase (*scale out*) or decrease (*scale in*) the number of instances. You can configure triggers to *fire* on any metric published to Amazon CloudWatch (for example, CPU utilization) and determine if the conditions you specified have been met. When the upper or lower thresholds of the conditions you have specified for the metric have been breached for the specified period of time, the trigger launches a long-running process called a *Scaling Activity*.

You can define a scaling trigger for your Elastic Beanstalk application using AWS Toolkit for Visual Studio.



Trigger Measurement:	NetworkOut	
Trigger Statistic:	Average	
Unit of Measurement:	Bytes	
Measurement Period (minutes):	5	(1 - 600)
Breach Duration (minutes):	5	(1 - 600)
Upper Threshold:	6000000	(0 - 20000000)
Upper Breach Scalement Increment:	1	
Lower Threshold:	2000000	(0 - 20000000)
Lower Breach Scalement Increment:	-1	

Amazon EC2 Auto Scaling triggers work by monitoring a specific Amazon CloudWatch metric of a particular instance. Metrics include CPU utilization, network traffic, and disk activity. Use the **Trigger Measurement** setting to select a metric for your trigger.

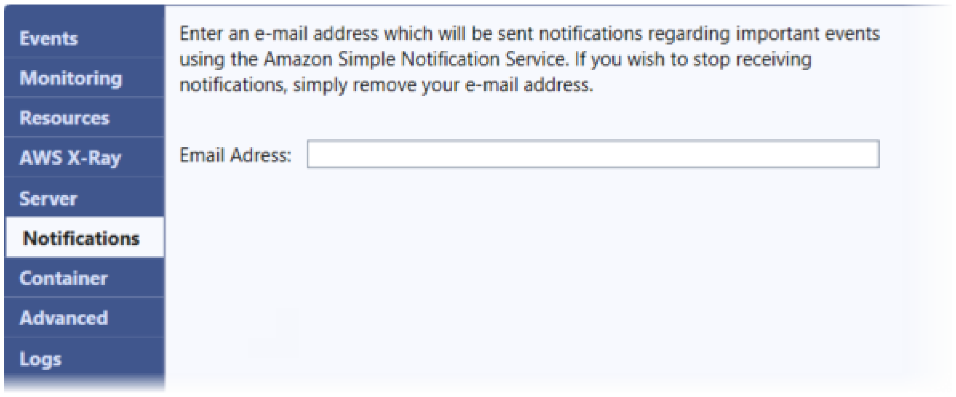
The following list describes the trigger parameters you can configure using the AWS Management Console.

- You can specify which statistic the trigger should use. You can select **Minimum**, **Maximum**, **Sum**, or **Average** for **Trigger Statistic**.
- For **Unit of Measurement**, specify the unit for the trigger measurement.
- The value in the **Measurement Period** box specifies how frequently Amazon CloudWatch measures the metrics for your trigger. The **Breach Duration** is the amount of time a metric can go beyond its defined limit (as specified for the **Upper Threshold** and **Lower Threshold**) before the trigger fires.
- For **Upper Breach Scale Increment** and **Lower Breach Scale Increment**, specify how many Amazon EC2 instances to add or remove when performing a scaling activity.

For more information on Amazon EC2 Auto Scaling, see the *Amazon EC2 Auto Scaling* section on Amazon Elastic Compute Cloud Documentation.

Configuring notifications using AWS toolkit for Visual Studio

Elastic Beanstalk uses the Amazon Simple Notification Service (Amazon SNS) to notify you of important events affecting your application. To enable Amazon SNS notifications, enter your email address in the **Email Address** box. To disable these notifications, remove your email address from the box.



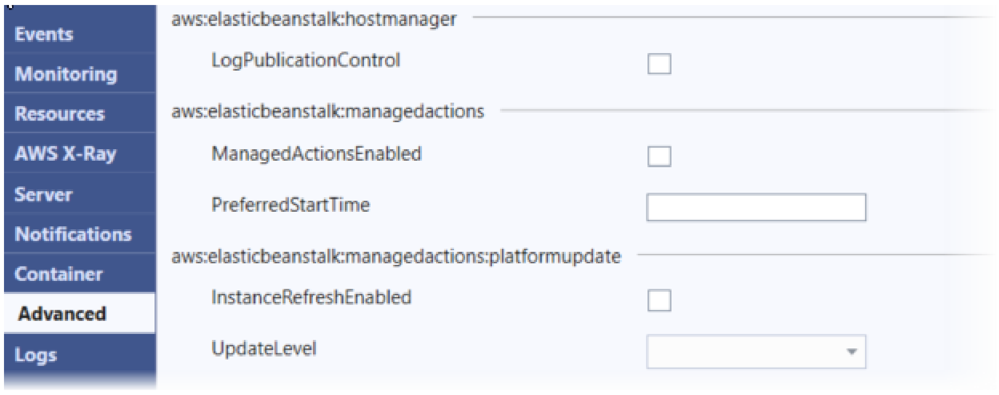
Enter an e-mail address which will be sent notifications regarding important events using the Amazon Simple Notification Service. If you wish to stop receiving notifications, simply remove your e-mail address.

Email Adress:

Configuring additional environment options using AWS toolkit for Visual Studio

Elastic Beanstalk defines a large number of configuration options that you can use to configure your environment's behavior and the resources that it contains. Configuration options are organized into namespaces like `aws:autoscaling:asg`. Each namespace defines options for an environment's Auto Scaling group. The **Advanced** panel lists the configuration option namespaces in alphabetical order that you can update after environment creation.

For a complete list of namespaces and options, including default and supported values for each, see General options for all environments and .NET Core on Linux platform options.



aws:elasticbeanstalk:hostmanager

LogPublicationControl

aws:elasticbeanstalk:managedactions

ManagedActionsEnabled

PreferredStartTime

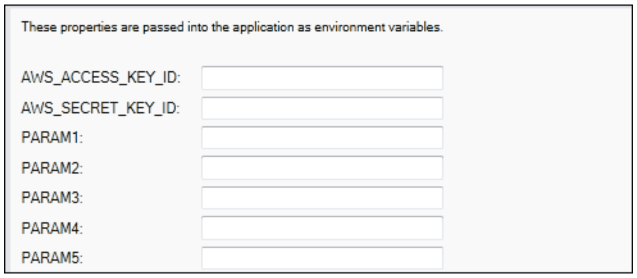
aws:elasticbeanstalk:managedactions:platformupdate

InstanceRefreshEnabled

UpdateLevel

Configuring .NET Core containers using the AWS toolkit for Visual Studio

The **Container** panel lets you specify environment variables that you can read from your application code.


These properties are passed into the application as environment variables.

AWS_ACCESS_KEY_ID:	
AWS_SECRET_KEY_ID:	
PARAM1:	
PARAM2:	
PARAM3:	
PARAM4:	
PARAM5:	

Monitoring application health

This topic explains how to monitor the health of your application's website. It's important to know that your production website is available and responding to requests. Elastic Beanstalk provides features to help you monitor your application's responsiveness. It monitors statistics about your application and alerts you when thresholds are exceeded.

For information about the health monitoring provided by Elastic Beanstalk, see Basic health reporting.

You can access operational information about your application by using either the AWSToolkit for Visual Studio or the AWS Management Console.

The toolkit displays your environment's status and application health in the **Status** field.

To monitor application health

1. In the AWS Toolkit for Visual Studio, in **AWS Explorer**, expand the Elastic Beanstalk node, and then expand your application node.
2. Open the context (right-click) menu for your application environment and select **View Status**.
3. On your application environment tab, select **Monitoring**.

   The **Monitoring** panel includes a set of graphs showing resource usage for your particular application environment.

   > **Note**
   >
   > By default, the time range is set to the last hour. To modify this setting, in the **Time Range** list, select a different time range.

You can use the AWS Toolkit for Visual Studio or the AWS Management Console to view events associated with your application.

To view application events

1. In the AWS Toolkit for Visual Studio, in **AWS Explorer**, expand the Elastic Beanstalk node and your application node.
2. Open the context (right-click) menu for your application environment and select **View Status**.
3. In your application environment tab, select **Events**.

Migrating from .NET on Windows Server platform to the .NET Core on Linux platform on Elastic Beanstalk

You can migrate applications that run on .NET on Windows Server platforms to the .NET Core on Linux platforms. Following are some considerations when migrating from Windows to Linux platforms.

Considerations for migrating to the .NET Core on Linux platform

Area	Changes and information
Applicati on configura tion	On Windows platforms, you use a deployment manifest to specify the applicati ons that run in your environment. The .NET Core on Linux platforms use a Procfile to specify the applications that run on your environment's instances. For details on bundling applications, see the section called "Bundling applications".
Proxy server	On Windows platforms, you use IIS as your application's proxy server. The .NET Core on Linux platforms include nginx as a reverse proxy by default. You can choose to use no proxy server and use Kestrel as your application's web server. To learn more, see the section called "Proxy server".
Routing	On Windows platforms, you use IIS in your application code and include a deployment manifest to configure the IIS path. For the .NET Core on Linux platform, you use ASP .NET Core routing in your application code, and update your environment's nginx configuration. To learn more, see the section called "Proxy server".

Area	Changes and information
Logs	The Linux and Windows platforms stream different logs. For details, see the section called “How Elastic Beanstalk sets up CloudWatch Logs”.

Deploying Go applications with Elastic Beanstalk

This chapter provides instructions for configuring and deploying your Go web application to AWS Elastic Beanstalk. Elastic Beanstalk makes it easy to deploy, manage, and scale your Go web applications using Amazon Web Services.

You can deploy your application in just a few minutes using the Elastic Beanstalk Command Line Interface (EB CLI) or by using the Elastic Beanstalk console. After you deploy your Elastic Beanstalk application, you can continue to use the EB CLI to manage your application and environment, or you can use the Elastic Beanstalk console, AWS CLI, or the APIs.

Follow the QuickStart for Go for step-by-step instructions to create and deploy a *Hello World* Go web application with the EB CLI.

Topics

QuickStart: Deploy a Go application to Elastic Beanstalk

This QuickStart tutorial walks you through the process of creating a Go application and deploying it to an AWS Elastic Beanstalk environment.

Note

Tutorial examples are intended for demonstration. Do not use the application for production traffic.

Sections

Your AWS account

If you're not already an AWS customer, you need to create an AWS account. Signing up enables you to access Elastic Beanstalk and other AWS services that you need.

If you already have an AWS account, you can move on to Prerequisites.

Create an AWS account

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.
2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.
2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Prerequisites

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

EB CLI

This tutorial uses the Elastic Beanstalk Command Line Interface (EB CLI). For details on installing and configuring the EB CLI, see Install EB CLI with setup script (recommended) and Configure the EB CLI.

Step 1: Create a Go application

Create a project directory.

```
~$ mkdir eb-go
~$ cd eb-go
```

Next, create an application that you'll deploy using Elastic Beanstalk. We'll create a "Hello World" RESTful web service.

This example prints a customized greeting that varies based on the path used to access the service.

Create a text file in this directory named `application.go` with the following contents.

Example ~/eb-go/application.go

```
package main
```

```
import (
 "fmt"
 "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
 if r.URL.Path == "/" {
  fmt.Fprintf(w, "Hello World! Append a name to the URL to say hello. For example, use
 %s/Mary to say hello to Mary.", r.Host)
 } else {
  fmt.Fprintf(w, "Hello, %s!", r.URL.Path[1:])
 }
}

func main() {
 http.HandleFunc("/", handler)
 http.ListenAndServe(":5000", nil)
}
```

Step 2: Deploy your Go application with the EB CLI

Next, you create your application environment and deploy your configured application with Elastic Beanstalk.

To create an environment and deploy your Go application

1. Initialize your EB CLI repository with the **eb init** command.

   ```
   ~/eb-go$ eb init -p go go-tutorial --region us-east-2
   Application go-tutorial has been created.
   ```

   This command creates an application named `go-tutorial` and configures your local repository to create environments with the latest Go platform version.

2. (Optional) Run **eb init** again to configure a default key pair so that you can use SSH to connect to the EC2 instance running your application.

   ```
   ~/eb-go$ eb init
   Do you want to set up SSH for your instances?
   (y/n): y
   Select a keypair.
   ```

```
1) my-keypair
2) [ Create new KeyPair ]
```

Select a key pair if you have one already, or follow the prompts to create one. If you don't see the prompt or need to change your settings later, run **eb init -i**.

3. Create an environment and deploy your application to it with **eb create**. Elastic Beanstalk automatically builds a zip file for your application and starts it on port 5000.

   ```
   ~/eb-go$ eb create go-env
   ```

   It takes about five minutes for Elastic Beanstalk to create your environment.

Step 3: Run your application on Elastic Beanstalk

When the process to create your environment completes, open your website with **eb open**.

```
~/eb-go$ eb open
```

Congratulations! You've deployed a Go application with Elastic Beanstalk! This opens a browser window using the domain name created for your application.

Step 4: Clean up

You can terminate your environment when you finish working with your application. Elastic Beanstalk terminates all AWS resources associated with your environment.

To terminate your Elastic Beanstalk environment with the EB CLI run the following command.

```
~/eb-go$ eb terminate
```

AWS resources for your application

You just created a single instance application. It serves as a straightforward sample application with a single EC2 instance, so it doesn't require load balancing or auto scaling. For single instance applications Elastic Beanstalk creates the following AWS resources:

- **EC2 instance** – An Amazon EC2 virtual machine configured to run web apps on the platform you choose.

Each platform runs a different set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination thereof. Most platforms use either Apache or nginx as a reverse proxy that processes web traffic in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow incoming traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic is not allowed on other ports.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name** – A domain name that routes to your web app in the form *`subdomain`.`region`.elasticbeanstalk.com*.

Elastic Beanstalk manages all of these resources. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Next steps

After you have an environment running an application, you can deploy a new version of the application or a different application at any time. Deploying a new application version is very quick because it doesn't require provisioning or restarting EC2 instances. You can also explore your new environment using the Elastic Beanstalk console. For detailed steps, see Explore your environment in the *Getting started* chapter of this guide.

After you deploy a sample application or two and are ready to start developing and running Go applications locally, see Setting up your Go development environment for Elastic Beanstalk.

Deploy with the Elastic Beanstalk console

You can also use the Elastic Beanstalk console to launch the sample application. For detailed steps, see Create an example application in the *Getting started* chapter of this guide.

Setting up your Go development environment for Elastic Beanstalk

This topic provides instructions to set up a Go development environment to test your application locally prior to deploying it to AWS Elastic Beanstalk. It also references websites that provide installation instructions for useful tools.

Installing Go

To run Go applications locally, install Go. If you don't need a specific version, get the latest version that Elastic Beanstalk supports. For a list of supported versions, see Go in the *AWS Elastic Beanstalk Platforms* document.

Download Go at https://golang.org/doc/install.

Installing the AWS SDK for Go

If you need to manage AWS resources from within your application, install the AWS SDK for Go by using the following command.

```
$ go get github.com/aws/aws-sdk-go
```

For more information, see AWS SDK for Go.

Using the Elastic Beanstalk Go platform

This topic describes how to configure, build, and run your Go applications on Elastic Beanstalk.

AWS Elastic Beanstalk supports a number of platform branches for different versions of the Go programming language. See Go in the *AWS Elastic Beanstalk Platforms* document for a full list.

For simple Go applications, there are two ways to deploy your application:

- Provide a source bundle with a source file at the root called `application.go` that contains the main package for your application. Elastic Beanstalk builds the binary using the following command:

  ```
  go build -o bin/application application.go
  ```

After the application is built, Elastic Beanstalk starts it on port 5000.

- Provide a source bundle with a binary file called `application`. The binary file can be located either at the root of the source bundle or in the `bin/` directory of the source bundle. If you place the `application` binary file in both locations, Elastic Beanstalk uses the file in the `bin/` directory.

  Elastic Beanstalk launches this application on port 5000.

In both cases, with our supported Go platform branches, you can also provide module requirements in a file called `go.mod`. For more information, see Migrating to Go Modules in the Go blog.

For more complex Go applications, there are two ways to deploy your application:

- Provide a source bundle that includes your application source files, along with a Buildfile and a Procfile. The Buildfile includes a command to build the application, and the Procfile includes instructions to run the application.
- Provide a source bundle that includes your application binary files, along with a Procfile. The Procfile includes instructions to run the application.

The Go platform includes a proxy server to serve static assets and forward traffic to your application. You can extend or override the default proxy configuration for advanced scenarios.

For details about the various ways you can extend an Elastic Beanstalk Linux-based platform, see the section called “Extending Linux platforms”.

Configuring your Go environment

The Go platform settings let you fine-tune the behavior of your Amazon EC2 instances. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration using the Elastic Beanstalk console.

Use the Elastic Beanstalk console to enable log rotation to Amazon S3 and configure variables that your application can read from the environment.

To configure your Go environment in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.

Log options

The Log Options section has two settings:

- **Instance profile** – Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3** – Specifies whether log files for your application's Amazon EC2 instances are copied to the Amazon S3 bucket associated with your application.

Static files

To improve performance, you can use the **Static files** section to configure the proxy server to serve static files (for example, HTML or images) from a set of directories inside your web application. For each directory, you set the virtual path to directory mapping. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application.

For details about configuring static files using configuration files or the Elastic Beanstalk console, see the section called "Static files".

Environment properties

The **Environment Properties** section lets you specify environment configuration settings on the Amazon EC2 instances that are running your application. Environment properties are passed in as key-value pairs to the application.

Inside the Go environment running in Elastic Beanstalk, environment variables are accessible using the `os.Getenv` function. For example, you could read a property named `API_ENDPOINT` to a variable with the following code:

```
endpoint := os.Getenv("API_ENDPOINT")
```

See Environment variables and other software settings for more information.

Go configuration namespace

You can use a configuration file to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be platform specific or apply to all platforms in the Elastic Beanstalk service as a whole. Configuration options are organized into *namespaces*.

The Go platform doesn't define any platform-specific namespaces. You can configure the proxy to serve static files by using the `aws:elasticbeanstalk:environment:proxy:staticfiles` namespace. For details and an example, see the section called "Static files".

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See Configuration options for more information.

The Amazon Linux AMI (preceding Amazon Linux 2) Go platform

If your Elastic Beanstalk Go environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Notes

- The information in this topic only applies to platform branches based on Amazon Linux AMI (AL1). AL2023/AL2 platform branches are incompatible with previous Amazon Linux AMI (AL1) platform versions and *require different configuration settings*.
- On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information about migrating to a current and fully supported Amazon Linux 2023 platform branch, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

Go configuration namespaces — Amazon Linux AMI (AL1)

You can use a configuration file to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be platform specific or apply to all platforms in the Elastic Beanstalk service as a whole. Configuration options are organized into *namespaces*.

Note

The information in this topic only applies to platform branches based on Amazon Linux AMI (AL1). AL2023/AL2 platform branches are incompatible with previous Amazon Linux AMI (AL1) platform versions and *require different configuration settings*.

The Amazon Linux AMI Go platform supports one platform-specific configuration namespace in addition to the namespaces supported by all platforms. The `aws:elasticbeanstalk:container:golang:staticfiles` namespace lets you define options that map paths on your web application to folders in your application source bundle that contain static content.

For example, this configuration file tells the proxy server to serve files in the `staticimages` folder at the path `/images`:

Example .ebextensions/go-settings.config

```
option_settings:
  aws:elasticbeanstalk:container:golang:staticfiles:
    /html: statichtml
    /images: staticimages
```

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See Configuration options for more information.

Configuring custom start commands with a Procfile on Elastic Beanstalk

To specify custom commands to start a Go application, include a file called `Procfile` at the root of your source bundle.

For details about writing and using a `Procfile`, see Buildfile and Procfile.

Example Procfile

```
web: bin/server
```

```
queue_process: bin/queue_processor
foo: bin/fooapp
```

You must call the main application `web`, and list it as the first command in your `Procfile`. Elastic Beanstalk exposes the main `web` application on the root URL of the environment; for example, `http://my-go-env.elasticbeanstalk.com`.

Elastic Beanstalk also runs any application whose name does not have the `web_` prefix, but these applications are not available from outside of your instance.

Elastic Beanstalk expects processes run from the `Procfile` to run continuously. Elastic Beanstalk monitors these applications and restarts any process that terminates. For short-running processes, use a Buildfile command.

Using a Procfile on Amazon Linux AMI (preceding Amazon Linux 2)

If your Elastic Beanstalk Go environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Notes

- The information in this topic only applies to platform branches based on Amazon Linux AMI (AL1). AL2023/AL2 platform branches are incompatible with previous Amazon Linux AMI (AL1) platform versions and *require different configuration settings*.
- On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information about migrating to a current and fully supported Amazon Linux 2023 platform branch, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

Port passing — Amazon Linux AMI (AL1)

Note

The information in this topic only applies to platform branches based on Amazon Linux AMI (AL1). AL2023/AL2 platform branches are incompatible with previous Amazon Linux AMI (AL1) platform versions and *require different configuration settings*.

Elastic Beanstalk configures the nginx proxy to forward requests to your application on the port number specified in the PORT environment property for your application. Your application should always listen on that port. You can access this variable within your application by calling the `os.Getenv("PORT")` method.

Elastic Beanstalk uses the port number specified in the PORT environment property for the port for the first application in the `Procfile`, and then increments the port number for each subsequent application in the `Procfile` by 100. If the PORT environment property is not set, Elastic Beanstalk uses 5000 for the initial port.

In the preceding example, the PORT environment property for the `web` application is 5000, the `queue_process` application is 5100, and the `foo` application is 5200.

You can specify the initial port by setting the PORT option with the aws:elasticbeanstalk:application:environment namespace, as shown in the following example.

```
option_settings:
  - namespace:  aws:elasticbeanstalk:application:environment
    option_name:  PORT
    value:  <first_port_number>
```

For more information about setting environment properties for your application, see Option settings.

Custom build and configuration with a Buildfile on Elastic Beanstalk

To specify a custom build and configuration command for your Go application, include a file called `Buildfile` at the root of your source bundle. The file name is case sensitive. Use the following format for the `Buildfile`:

```
<process_name>: <command>
```

The command in your `Buildfile` must match the following regular expression: `^[A-Za-z0-9_]+:\s*.+$`.

Elastic Beanstalk doesn't monitor the application that is run with a `Buildfile`. Use a `Buildfile` for commands that run for short periods and terminate after completing their tasks. For long-running application processes that should not exit, use the Procfile instead.

In the following example of a `Buildfile`, `build.sh` is a shell script that is located at the root of the source bundle:

```
make: ./build.sh
```

All paths in the `Buildfile` are relative to the root of the source bundle. If you know in advance where the files reside on the instance, you can include absolute paths in the `Buildfile`.

Configuring the proxy server

Elastic Beanstalk uses nginx as the reverse proxy to map your application to your Elastic Load Balancing load balancer on port 80. Elastic Beanstalk provides a default nginx configuration that you can either extend or override completely with your own configuration.

By default, Elastic Beanstalk configures the nginx proxy to forward requests to your application on port 5000. You can override the default port by setting the `PORT` environment property to the port on which your main application listens.

Note

The port that your application listens on doesn't affect the port that the nginx server listens to receive requests from the load balancer.

Configuring the proxy server on your platform version

All AL2023/AL2 platforms support a uniform proxy configuration feature. For more information about configuring the proxy server on your platform versions running AL2023/AL2, see Reverse proxy configuration.

Configuring the proxy on Amazon Linux AMI (preceding Amazon Linux 2)

Notes

- The information in this topic only applies to platform branches based on Amazon Linux AMI (AL1). AL2023/AL2 platform branches are incompatible with previous Amazon Linux AMI (AL1) platform versions and *require different configuration settings*.
- On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information about migrating to a current

and fully supported Amazon Linux 2023 platform branch, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

If your Elastic Beanstalk Go environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the information in this section.

Extending and overriding the default proxy configuration — Amazon Linux AMI (AL1)

Elastic Beanstalk uses nginx as the reverse proxy to map your application to your load balancer on port 80. If you want to provide your own nginx configuration, you can override the default configuration provided by Elastic Beanstalk by including the `.ebextensions/nginx/nginx.conf` file in your source bundle. If this file is present, Elastic Beanstalk uses it in place of the default nginx configuration file.

If you want to include directives in addition to those in the `nginx.conf` `http` block, you can also provide additional configuration files in the `.ebextensions/nginx/conf.d/` directory of your source bundle. All files in this directory must have the `.conf` extension.

To take advantage of functionality provided by Elastic Beanstalk, such as Enhanced health reporting and monitoring in Elastic Beanstalk, automatic application mappings, and static files, you must include the following line in the `server` block of your nginx configuration file:

```
include conf.d/elasticbeanstalk/*.conf;
```

Deploying Java applications with Elastic Beanstalk

This chapter provides instructions for configuring and deploying your Java applications to AWS Elastic Beanstalk. Elastic Beanstalk makes it easy to deploy, manage, and scale your Java web applications using Amazon Web Services.

You can deploy your application in just a few minutes using the Elastic Beanstalk Command Line Interface (EB CLI) or by using the Elastic Beanstalk console. After you deploy your Elastic Beanstalk application, you can continue to use the EB CLI to manage your application and environment, or you can use the Elastic Beanstalk console, AWS CLI, or the APIs.

Follow the QuickStart for Java for step-by-step instructions to create and deploy a *Hello World* Java web application with the EB CLI. If you're interested in step-by-step instructions to create a simple *Hello World* Java JSP application to deploy with the EB CLI to our Tomcat based platform, try the QuickStart for Java on Tomcat.

The Java platform branches

AWS Elastic Beanstalk supports two platforms for Java applications.

- **Tomcat –** A platform based on *Apache Tomcat*, an open source web container for applications that use Java servlets and JavaServer Pages (JSPs) to serve HTTP requests. Tomcat facilitates web application development by providing multithreading, declarative security configuration, and extensive customization. Elastic Beanstalk has platform branches for each of Tomcat's current major versions. For more information, see The Tomcat platform.
- **Java SE –** A platform for applications that don't use a web container, or use one other than Tomcat, such as Jetty or GlassFish. You can include any library Java Archives (JARs) used by your application in the source bundle that you deploy to Elastic Beanstalk. For more information, see The Java SE platform.

Recent branches of both the Tomcat and Java SE platforms are based on Amazon Linux 2 and later, and use *Corretto*—the AWS Java SE distribution. The names of these platform branches include the word *Corretto* instead of *Java*.

For a list of current platform versions, see Tomcat and Java SE in the *AWS Elastic Beanstalk Platforms* guide.

AWS tools

AWS provides several tools for working with Java and Elastic Beanstalk. Regardless of the platform branch that you choose, you can use the AWS SDK for Java to use other AWS services from within your Java application. The AWS SDK for Java is a set of libraries that allow you to use AWS APIs from your application code without writing the raw HTTP calls from scratch.

If you prefer to manage your applications from the command line, install the Elastic Beanstalk Command Line Interface (EB CLI) and use it to create, monitor, and manage your Elastic Beanstalk environments. If you run multiple environments for your application, the EB CLI integrates with Git to let you associate each of your environments with a different Git branch.

Topics

QuickStart: Deploy a Java application to Elastic Beanstalk

This QuickStart tutorial walks you through the process of creating a Java application and deploying it to an AWS Elastic Beanstalk environment.

Note

Tutorial examples are intended for demonstration. Do not use the application for production traffic.

Sections

Your AWS account

If you're not already an AWS customer, you need to create an AWS account. Signing up enables you to access Elastic Beanstalk and other AWS services that you need.

If you already have an AWS account, you can move on to Prerequisites.

Create an AWS account

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.
2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.
2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Prerequisites

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

EB CLI

This tutorial uses the Elastic Beanstalk Command Line Interface (EB CLI). For details on installing and configuring the EB CLI, see Install EB CLI with setup script (recommended)and Configure the EB CLI.

Java and Maven

If you don't have Amazon Corretto installed on your local machine, you can install it by following the installation instructions in the *Amazon Corretto User Guide*.

Verify your Java installation by running the following command.

```
~$ java -version
```

This tutorial uses Maven. Follow the download and installation instructions on the Apache Maven Project website. For more information about Maven see the Maven Users Centre on the Apache Maven Project website.

Verify your Maven installation by running the following command.

```
~$ mvn -v
```

Step 1: Create a Java application

Create a project directory.

```
~$ mkdir eb-java
~$ cd eb-java
```

Next, create an application that you'll deploy using Elastic Beanstalk. We'll create a "Hello World" RESTful web service.

This example uses the Spring Boot framework. This application opens a listener on port 5000. Elastic Beanstalk forward requests to your application on port 5000 by default.

Create the following files:

This file creates a simple Spring Boot application.

Example ~/eb-java/src/main/java/com/example/Application.java

```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

This file creates a mapping that returns a String that we define here.

Example ~/eb-java/src/main/java/com/example/Controller.java

```
package com.example;
```

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class Controller {

    @GetMapping("/")
    public String index() {
       return "Hello Elastic Beanstalk!";
    }
}
```

This file defines the Maven project configuration.

Example `~/eb-java/pom.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.3</version>
  </parent>

  <groupId>com.example</groupId>
  <artifactId>BeanstalkJavaExample</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <java.version>21</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-rest</artifactId>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>

</project>
```

This properties file overrides the default port to be 5000. This is the default port that Elastic Beanstalk sends traffic to for Java applications.

Example `~/eb-java/application.properties`

```
server.port=5000
```

Step 2: Run your application locally

Package your application with the following command:

```
~/eb-java$ mvn clean package
```

Run your application locally with the following command:

```
~/eb-java$ java -jar target/BeanstalkJavaExample-1.0-SNAPSHOT.jar
```

While the application is running, navigate to `http://127.0.0.1:5000/` in your browser. You should see the text “Hello Elastic Beanstalk!”.

Step 3: Deploy your Java application with the EB CLI

Before deploying your Java application to Elastic Beanstalk, let’s clean the build application from your directory and create a Buildfile and a Procfile to control how the application is built and run on your Elastic Beanstalk environment.

To prepare and configure for application deployment

1. Clean the built application.

```
~/eb-java$ mvn clean
```

2. Create your `Buildfile`.

   Example ~/eb-java/Buildfile

   ```
   build: mvn clean package
   ```

   This `Buildfile` specifies the command used to build your application. If you don't include a `Buildfile` for a Java application, Elastic Beanstalk doesn't attempt to build your application.

3. Create your `Procfile`.

   Example ~/eb-java/Procfile

   ```
   web: java -jar target/BeanstalkJavaExample-1.0-SNAPSHOT.jar
   ```

   This `Procfile` specifies the command used to run your application. If you don't include a `Procfile` for a Java application, Elastic Beanstalk assumes there is one JAR file in the root of your source bundle and tries to run it with the `java -jar` command.

Now that you have set up the configuration files to build and start your application, you're ready to deploy it.

To create an environment and deploy your Java application

1. Initialize your EB CLI repository with the **eb init** command.

   ```
   ~/eb-java eb init -p corretto java-tutorial --region us-east-2

   Application java-tutorial has been created.
   ```

   This command creates an application named `java-tutorial` and configures your local repository to create environments with the latest Java platform version.

2. (Optional) Run **eb init** again to configure a default key pair so that you can use SSH to connect to the EC2 instance running your application.

   ```
   ~/eb-java$ eb init
   ```

```
Do you want to set up SSH for your instances?
(y/n): y
Select a keypair.
1) my-keypair
2) [ Create new KeyPair ]
```

Select a key pair if you have one already, or follow the prompts to create one. If you don't see the prompt or need to change your settings later, run **eb init -i**.

3. Create an environment and deploy your application to it with **eb create**. Elastic Beanstalk automatically builds a zip file for your application and starts it on port 5000.

```
~/eb-java$ eb create java-env
```

It takes about five minutes for Elastic Beanstalk to create your environment.

Step 4: Run your application on Elastic Beanstalk

When the process to create your environment completes, open your website with **eb open**.

```
~/eb-java eb open
```

Congratulations! You've deployed a Java application with Elastic Beanstalk! This opens a browser window using the domain name created for your application.

Step 5: Clean up

You can terminate your environment when you finish working with your application. Elastic Beanstalk terminates all AWS resources associated with your environment.

To terminate your Elastic Beanstalk environment with the EB CLI run the following command.

```
~/eb-java$ eb terminate
```

AWS resources for your application

You just created a single instance application. It serves as a straightforward sample application with a single EC2 instance, so it doesn't require load balancing or auto scaling. For single instance applications Elastic Beanstalk creates the following AWS resources:

- **EC2 instance** – An Amazon EC2 virtual machine configured to run web apps on the platform you choose.

  Each platform runs a different set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination thereof. Most platforms use either Apache or nginx as a reverse proxy that processes web traffic in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group** – An Amazon EC2 security group configured to allow incoming traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic is not allowed on other ports.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name** – A domain name that routes to your web app in the form *subdomain*.*region*.*elasticbeanstalk.com*.

Elastic Beanstalk manages all of these resources. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Next steps

After you have an environment running an application, you can deploy a new version of the application or a different application at any time. Deploying a new application version is very quick because it doesn't require provisioning or restarting EC2 instances. You can also explore your new environment using the Elastic Beanstalk console. For detailed steps, see Explore your environment in the *Getting started* chapter of this guide.

Try more tutorials

If you'd like to try other tutorials with different example applications, see Sample applications and tutorials.

After you deploy a sample application or two and are ready to start developing and running Java applications locally, see Setting up your Java development environment.

Deploy with the Elastic Beanstalk console

You can also use the Elastic Beanstalk console to launch the sample application. For detailed steps, see Create an example application in the *Getting started* chapter of this guide.

QuickStart: Deploy a Java JSP web application for Tomcat to Elastic Beanstalk

This tutorial walks you through the process of creating a simple Java web application using JavaServer Pages (JSPs). If you'd like to bundle multiple web applications in the form of WAR files in a single Elastic Beanstalk environment, see Bundling multiple WAR files for Tomcat environments.

Note

Tutorial examples are intended for demonstration. Do not use the application for production traffic.

Sections

Your AWS account

If you're not already an AWS customer, you need to create an AWS account. Signing up enables you to access Elastic Beanstalk and other AWS services that you need.

If you already have an AWS account, you can move on to Prerequisites.

Create an AWS account

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.
2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Prerequisites

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

EB CLI

This tutorial uses the Elastic Beanstalk Command Line Interface (EB CLI). For details on installing and configuring the EB CLI, see Install EB CLI with setup script (recommended) and Configure the EB CLI.

Step 1: Create a Java JSP application

Create a project directory.

```
~$ mkdir eb-tomcat
~$ cd eb-tomcat
```

Next, create an application that you'll deploy using Elastic Beanstalk. We'll create a "Hello World" web application.

Create a simple JSP file named `index.jsp`.

Example ~/eb-tomcat/index.jsp

```
<html>
  <body>
    <%out.println("Hello Elastic Beanstalk!");%>
  </body>
</html>
```

Step 2: Deploy your Java JSP application with the EB CLI

Run the following commands to create an Elastic Beanstalk environment for this application.

To create an environment and deploy your Java JSP application

1. Initialize your EB CLI repository with the **eb init** command.

   ```
   ~/eb-tomcat$ eb init -p tomcat tomcat-tutorial --region us-east-2
   ```

   This command creates an application named `tomcat-tutorial` and configures your local repository to create environments with the latest Tomcat platform version.

2. (Optional) Run **eb init** again to configure a default key pair so that you can use SSH to connect to the EC2 instance running your application.

   ```
   ~/eb-go$ eb init
   Do you want to set up SSH for your instances?
   (y/n): y
   Select a keypair.
   1) my-keypair
   2) [ Create new KeyPair ]
   ```

   Select a key pair if you have one already, or follow the prompts to create one. If you don't see the prompt or need to change your settings later, run **eb init -i**.

3. Create an environment and deploy your application to it with **eb create**. Elastic Beanstalk automatically builds a zip file for your application and starts it on port 5000.

   ```
   ~/eb-tomcat$ eb create tomcat-env
   ```

   It takes about five minutes for Elastic Beanstalk to create your environment.

Step 3: Run your application on Elastic Beanstalk

When the process to create your environment completes, open your website with **eb open**.

```
~/eb-tomcat$ eb open
```

Congratulations! You've deployed a Java JSP application with Elastic Beanstalk! This opens a browser window using the domain name created for your application.

Step 4: Clean up

You can terminate your environment when you finish working with your application. Elastic Beanstalk terminates all AWS resources associated with your environment.

To terminate your Elastic Beanstalk environment with the EB CLI run the following command.

```
~/eb-tomcat$ eb terminate
```

AWS resources for your application

You just created a single instance application. It serves as a straightforward sample application with a single EC2 instance, so it doesn't require load balancing or auto scaling. For single instance applications Elastic Beanstalk creates the following AWS resources:

- **EC2 instance** – An Amazon EC2 virtual machine configured to run web apps on the platform you choose.

  Each platform runs a different set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination thereof. Most platforms use either Apache or nginx as a reverse proxy that processes web traffic in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group** – An Amazon EC2 security group configured to allow incoming traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic is not allowed on other ports.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name** – A domain name that routes to your web app in the form *subdomain.region.elasticbeanstalk.com*.

Elastic Beanstalk manages all of these resources. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Next steps

After you have an environment running an application, you can deploy a new version of the application or a different application at any time. Deploying a new application version is very quick because it doesn't require provisioning or restarting EC2 instances. You can also explore your new environment using the Elastic Beanstalk console. For detailed steps, see Explore your environment in the *Getting started* chapter of this guide.

Try more tutorials

If you'd like to try other tutorials with different example applications, see Sample applications and tutorials.

After you deploy a sample application or two and are ready to start developing and running Java applications in a local Tomcat web container, see Setting up your Java development environment.

Deploy with the Elastic Beanstalk console

You can also use the Elastic Beanstalk console to launch the sample application. For detailed steps, see Create an example application in the *Getting started* chapter of this guide.

Setting up your Java development environment

This topic provides instructions to set up a Java development environment to test your application locally prior to deploying it to AWS Elastic Beanstalk. It also references websites that provide installation instructions for useful tools.

Sections

Installing the Java development kit

Install the Java Development Kit (JDK). If you don't have a preference, get the latest version. Download the JDK at oracle.com

The JDK includes the Java compiler, which you can use to build your source files into class files that can be executed on an Elastic Beanstalk web server.

Installing a web container

If you don't already have another web container or framework, install a version of Tomcat that Elastic Beanstalk supports for your Amazon Linux operating system. For a list of the current versions of Apache Tomcat that Elastic Beanstalk supports, see Tomcat in the *AWS Elastic Beanstalk Platforms* document. Download the Tomcat version that applies to your environment from the Apache Tomcat website.

Downloading libraries

Elastic Beanstalk platforms include few libraries by default. Download libraries that your application will use and save them in your project folder to deploy in your application source bundle.

If you've installed Tomcat locally, you can copy the servlet API and JavaServer Pages (JSP) API libraries from the installation folder. If you deploy to a Tomcat platform version, you don't need to include these files in your source bundle, but you do need to have them in your `classpath` to compile any classes that use them.

JUnit, Google Guava, and Apache Commons provide several useful libraries. Visit their home pages to learn more:

- Download JUnit
- Download Google Guava
- Download Apache Commons

Installing the AWS SDK for Java

If you need to manage AWS resources from within your application, install the AWS SDK for Java. For example, with the AWS SDK for Java, you can use Amazon DynamoDB (DynamoDB) to share session states of Apache Tomcat applications across multiple web servers. For more information, see Manage Tomcat Session State with Amazon DynamoDB in the AWS SDK for Java documentation.

Visit the AWS SDK for Java home page for more information and installation instructions.

Installing an IDE or text editor

Integrated development environments (IDEs) provide a wide range of features that facilitate application development. If you haven't used an IDE for Java development, try Eclipse and IntelliJ and see which works best for you.

- Install Eclipse IDE for Java EE Developers
- Install IntelliJ

An IDE might add files to your project folder that you might not want to commit to source control. To prevent committing these files to source control, use `.gitignore` or your source control tool's equivalent.

If you just want to begin coding and don't need all of the features of an IDE, consider installing Sublime Text.

Note

On May 31, 2023, the AWS Toolkit for Eclipse reached end of life and is no longer supported by AWS. For additional details regarding the end of life cycle for the AWS Toolkit for Eclipse, see the README.md file on the AWS Toolkit for Eclipse GitHub repository.

More Elastic Beanstalk example applications and tutorials for Java

This section provides additional applications and tutorials. The QuickStart for Java and QuickStart for Java on Tomcat topics located earlier in this topic walk you through launching a sample Java application with the EB CLI.

To get started with Java applications on AWS Elastic Beanstalk, all you need is an application source bundle to upload as your first application version and to deploy to an environment. When you create an environment, Elastic Beanstalk allocates all of the AWS resources needed to run a scalable web application.

Launching an environment with a sample Java application

Elastic Beanstalk provides single page sample applications for each platform as well as more complex examples that show the use of additional AWS resources such as Amazon RDS and language or platform-specific features and APIs.

The single page samples are the same code that you get when you create an environment without supplying your own source code. The more complex examples are hosted on GitHub and may need to be compiled or built prior to deploying to an Elastic Beanstalk environment.

Samples

Nam	Supported versions	Envir nt type	Source	Description
Tomc (singl page)	All *Tomcat with Corretto* platform branches	Web Serve Worke	tomcat.zi p	Tomcat web application with a single page (`index.jsp`) configured to be displayed at the website root. For worker environments, this sample includes a `cron.yaml` file that configures a scheduled task that calls `scheduled.jsp` once per minute. When `scheduled.jsp` is called, it writes to a log file at `/tmp/sample-app.log` . Finally, a configuration file is included in `.ebextensions` that copies the logs from `/tmp/` to the locations read by Elastic Beanstalk when you request environment logs. If you enable X-Ray integration on an environment running this sample, the application shows additional content regarding X-Ray and provides an option to

Nam	Supported versions	Enviro nt type	Source	Description
				generate debug information that you can view in the X-Ray console.
Corre (singl page)	Corretto 11 Corretto 8	Web Serve	corretto. zip	Corretto application with `Buildfile` and `Procfile` configuration files. If you enable X-Ray integration on an environment running this sample, the application shows additional content regarding X-Ray and provides an option to generate debug information that you can view in the X-Ray console.

Nam	Supported versions	Enviro nt type	Source	Description
Score	Java 8	Web Serve	[Clone the repo at GitHub.co m](#)	*Scorekeep* is a RESTful web API that uses the Spring framework to provide an interface for creating and managing users, sessions, and games. The API is bundles with an Angular 1.5 web app that consumes the API over HTTP. The application uses features of the Java SE platform to download dependencies and build on-instance, minimizing the size of the souce bundle. The application also includes nginx configuration files that override the default configuration to serve the frontend web app statically on port 80 through the proxy, and route requests to paths under `/api` to the API running on `localhost:5000` . Scorekeep also includes an `xray` branch that shows how to instrument a Java application for use with AWS X-Ray. It shows instrumentation of incoming HTTP requests with a servlet filter, automatic and manual AWS SDK client instrumentation, recorder configuration, and instrumentation of outgoing HTTP requests and SQL clients. See the readme for instructions or use the [AWS X-Ray getting started tutorial](#) to try the application with X-Ray.

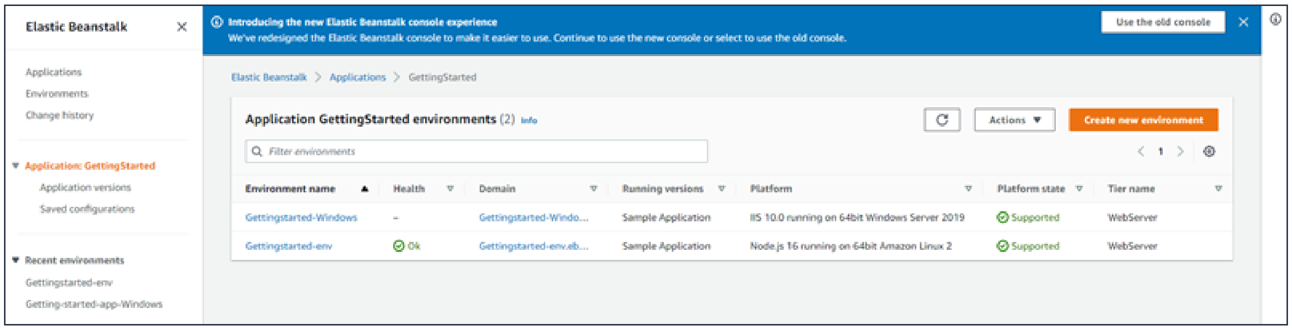
Nam	Supported versions	Enviro nt type	Source	Description
Does it Have Snak	Tomcat 8 with Java 8	Web Serve	[Clone the repo at GitHub.com]()	*Does it Have Snakes?* is a Tomcat web application that shows the use of Elastic Beanstalk configuration files, Amazon RDS, JDBC, PostgreSQL, Servlets, JSPs, Simple Tag Support, Tag Files, Log4J, Bootstrap, and Jackson. The source code for this project includes a minimal build script that compiles the servlets and models into class files and packages the required files into a Web Archive that you can deploy to an Elastic Beanstalk environment. See the readme file in the project repository for full instructions.
Locu Load Gene	Java 8	Web Serve	[Clone the repo at GitHub.com]()	Web application that you can use to load test another web application running in a different Elastic Beanstalk environment. Shows the use of `Buildfile` and `Procfile` files, DynamoDB, and [Locust](), an open source load testing tool.

Download any of the sample applications and deploy it to Elastic Beanstalk by following these steps:

To launch an environment with a sample application (console)

1. Open the [Elastic Beanstalk console](), and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**. Select an existing application in the list. You can also choose to create one, following the instructions in [Managing applications]() .
3. On the application overview page, choose **Create new environment**.

The following image displays the application overview page.



Introducing the new Elastic Beanstalk console experience
We've redesigned the Elastic Beanstalk console to make it easier to use. Continue to use the new console or select to use the old console.

Application GettingStarted environments (2) Info

Environment name	Health	Domain	Running versions	Platform	Platform state	Tier name
Gettingstarted-Windows	-	Gettingstarted-Windo...	Sample Application	IIS 10.0 running on 64bit Windows Server 2019	✓ Supported	WebServer
Gettingstarted-env	✓ Ok	Gettingstarted-env.eb...	Sample Application	Node.js 16 running on 64bit Amazon Linux 2	✓ Supported	WebServer

This launches the **Create environment** wizard. The wizard provides a set of steps for you to create a new environment.

4. For **Environment tier**, choose the **Web server environment** or **Worker environment** environment tier. You can't change an environment's tier after creation.

> **Note**
>
> The .NET on Windows Server platform doesn't support the worker environment tier.

The **Application information** fields default, based on the application that you previously chose.

In the **Environment information** grouping the **Environment name** defaults, based on the application name. If you prefer a different environment name you can enter another value in the field. You can optionally enter a **Domain** name; otherwise Elastic Beanstalk autogenerates a value. You can also optionally enter an **Environment description.**

5. For **Platform**, select the platform and platform branch that match the language your application uses.

> **Note**
>
> Elastic Beanstalk supports multiple versions for most of the platforms that are listed. By default, the console selects the recommended version for the platform and platform branch you choose. If your application requires a different version, you can select it here. For information about supported platform versions, see the section called "Supported platforms".

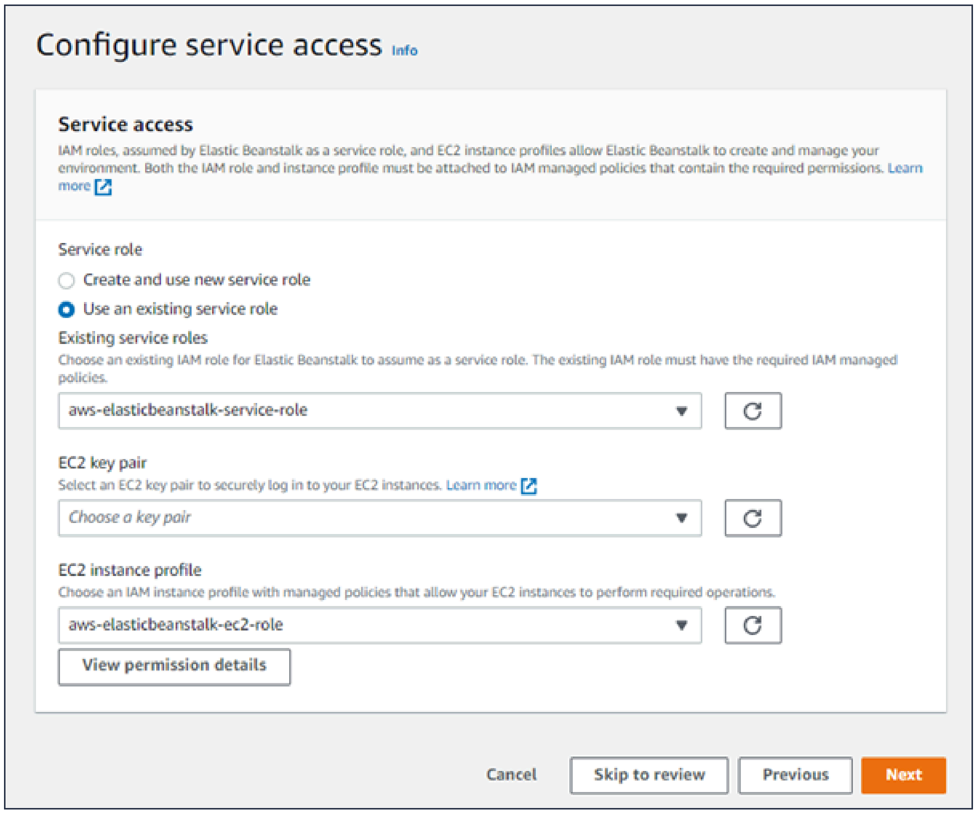
6. For **Application code**, you have some choices for launching a sample application.
   - To launch the default sample application without supplying the source code, choose **Sample application**. This action chooses the single page application that Elastic Beanstalk provides for the platform you previously selected.
   - If you downloaded a sample application from this guide or another source, do the following steps.

     a. Select **Upload your code**.

     b. Next choose **Local file**, then under **Upload application**, select **Choose file**.

     c. Your computer's operating system will present you with an interface to select the local file that you downloaded. Select the source bundle file and continue.
7. For **Presets**, choose **Single instance**.
8. Choose **Next**.
9. The **Configure service access** page displays.

   The following image illustrates the **Configure service access** page.



Configure service access Info

Service access

IAM roles, assumed by Elastic Beanstalk as a service role, and EC2 instance profiles allow Elastic Beanstalk to create and manage your environment. Both the IAM role and instance profile must be attached to IAM managed policies that contain the required permissions. Learn more

Service role

Create and use new service role [✓ Use an existing service role]

Existing service roles

Choose an existing IAM role for Elastic Beanstalk to assume as a service role. The existing IAM role must have the required IAM managed policies.

aws-elasticbeanstalk-service-role

EC2 key pair

Select an EC2 key pair to securely log in to your EC2 instances. Learn more

Choose a key pair

EC2 instance profile

Choose an IAM instance profile with managed policies that allow your EC2 instances to perform required operations.

aws-elasticbeanstalk-ec2-role

10. Choose a value from the **Existing Service Roles** dropdown.

11. (Optional) If you previously created an EC2 key pair, you can select it from the **EC2 key pair** field dropdown. You would use it to securely log in to the Amazon EC2 instance that Elastic Beanstalk provisions for your application. If you skip this step, you can always create and assign an EC2 key pair after the environment is created. For more information, see EC2 key pair.

12. Next, we'll focus on the **EC2 instance profile** dropdown list. The values displayed in this dropdown list may vary, depending on whether you account has previously created a new environment.

    Choose one of the following items, based on the values displayed in your list.

    - If `aws-elasticbeanstalk-ec2-role` displays in the dropdown list, select it from the dropdown list.
    - If another value displays in the list, and it's the default EC2 instance profile intended for your environments, select it from the dropdown list.
    - If the **EC2 instance profile** dropdown list doesn't list any values, you'll need to create an instance profile.

      Create an instance profile

      To create an instance profile, we'll take a detour to another procedure on this same page. Go to the end of this procedure and expand the procedure that follows, *Create IAM Role for EC2 instance profile*.
      Complete the steps in *Create IAM Role for EC2 instance profile* to create an IAM Role that you can subsequently select for the EC2 instance profile. Then return back to this step.

      Now that you've created an IAM Role, and refreshed the list, it displays as a choice in the dropdown list. Select the IAM Role you just created from the **EC2 instance profile** dropdown list.

13.
    Choose **Skip to Review** on the **Configure service access** page.

    This will select the default values for this step and skip the optional steps.

14. The **Review** page displays a summary of all your choices.

To further customize your environment, choose **Edit** next to the step that includes any items you want to configure. You can set the following options only during environment creation:

- Environment name
- Domain name
- Platform version
- Processor
- VPC
- Tier

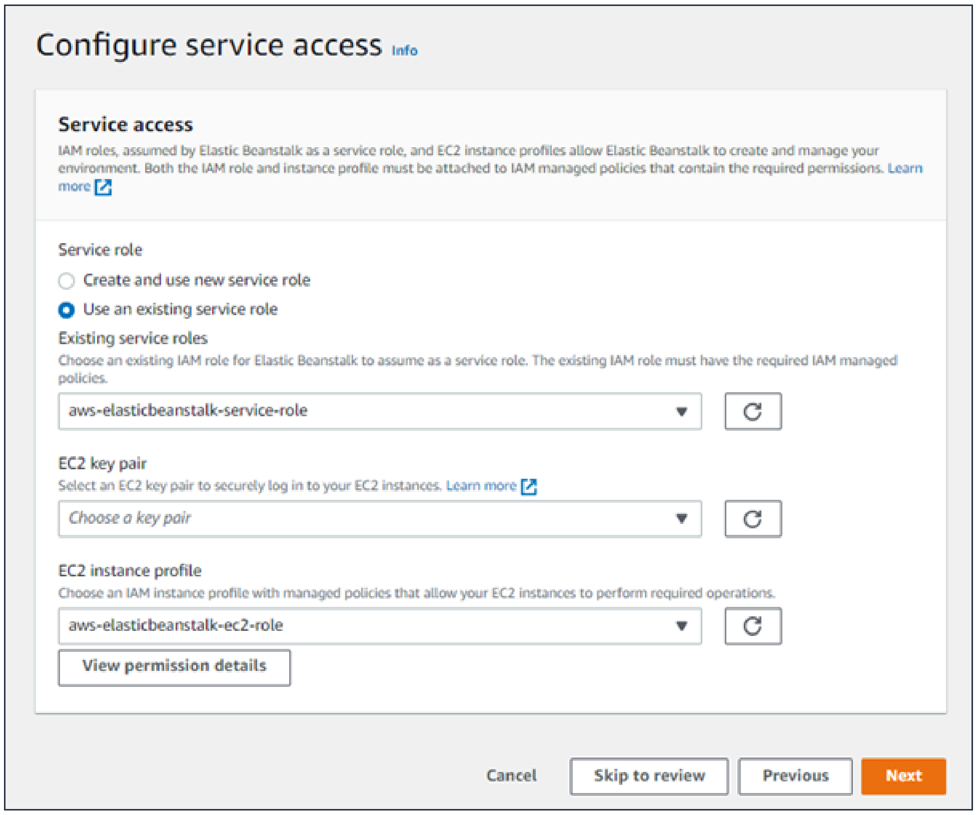
You can change the following settings after environment creation, but they require new instances or other resources to be provisioned and can take a long time to apply:

- Instance type, root volume, key pair, and AWS Identity and Access Management (IAM) role
- Internal Amazon RDS database
- Load balancer

For details on all available settings, see The create new environment wizard.

15. Choose **Submit** at the bottom of the page to initialize the creation of your new environment.

Create IAM Role for EC2 instance profile



Configure service access Info

Service access

IAM roles, assumed by Elastic Beanstalk as a service role, and EC2 instance profiles allow Elastic Beanstalk to create and manage your environment. Both the IAM role and instance profile must be attached to IAM managed policies that contain the required permissions. Learn more

Service role

- Create and use new service role
- [✓ Use an existing service role]

Existing service roles

Choose an existing IAM role for Elastic Beanstalk to assume as a service role. The existing IAM role must have the required IAM managed policies.

aws-elasticbeanstalk-service-role

EC2 key pair

Select an EC2 key pair to securely log in to your EC2 instances. Learn more

Choose a key pair

EC2 instance profile

Choose an IAM instance profile with managed policies that allow your EC2 instances to perform required operations.

aws-elasticbeanstalk-ec2-role

To create the EC2 instance profile

1. Choose **Create role**.
2. For **Trusted entity type**, choose **AWS service**.
3. For **Use case**, choose **Elastic Beanstalk – Compute**.
4. Choose **Next**.
5. Verify that **Permissions policies** include the following, then choose **Next**:
   - `AWSElasticBeanstalkWebTier`
   - `AWSElasticBeanstalkWorkerTier`
   - `AWSElasticBeanstalkMulticontainerDocker`
6. Choose **Create role**.
7. Return to the **Configure service access** tab, refresh the list, then select the newly created EC2 instance profile.

Next steps

After you have an environment running an application, you can deploy a new version of the application or a completely different application at any time. Deploying a new application version is very quick because it doesn't require provisioning or restarting EC2 instances.

After you've deployed a sample application or two and are ready to start developing and running Java applications locally, see the next section to set up a Java development environment with all of the tools and libraries that you will need.

Using the Elastic Beanstalk Tomcat platform

This topic describes how to configure, build, and run your Java applications that run on the Elastic Beanstalk Tomcat platform.

The AWS Elastic Beanstalk Tomcat platform is a set of platform versions for Java web applications that can run in a Tomcat web container. Tomcat runs behind an nginx proxy server. Each platform branch corresponds to a major version of Tomcat.

Configuration options are available in the Elastic Beanstalk console for modifying the configuration of a running environment. To avoid losing your environment's configuration when you terminate it, you can use saved configurations to save your settings and later apply them to another environment.

To save settings in your source code, you can include configuration files. Settings in configuration files are applied every time you create an environment or deploy your application. You can also use configuration files to install packages, run scripts, and perform other instance customization operations during deployments.

The Elastic Beanstalk Tomcat platform includes a reverse proxy that forwards requests to your application. You can use configuration options to configure the proxy server to serve static assets from a folder in your source code to reduce the load on your application. For advanced scenarios, you can include your own `.conf` files in your source bundle to extend the Elastic Beanstalk proxy configuration or overwrite it completely.

Note

Elastic Beanstalk supports nginx (the default) and Apache HTTP Server as the proxy servers on the Tomcat platform. If your Elastic Beanstalk Tomcat environment uses an Amazon Linux AMI platform branch (preceding Amazon Linux 2), you also have the option of using

Apache HTTP Server Version 2.2. Apache (latest) is the default on these older platform branches.
On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information about migrating to a current and fully supported Amazon Linux 2023 platform branch, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

You must package Java applications in a web application archive (WAR) file with a specific structure. For information on the required structure and how it relates to the structure of your project directory, see Structuring your project folder.

To run multiple applications on the same web server, you can bundle multiple WAR files into a single source bundle. Each application in a multiple WAR source bundle runs at the root path (`ROOT.war` runs at *`myapp`*`.elasticbeanstalk.com/`) or at a path directly beneath it (`app2.war` runs at *`myapp`*`.elasticbeanstalk.com/`*`app2`*`/`), as determined by the name of the WAR. In a single WAR source bundle, the application always runs at the root path.

Settings applied in the Elastic Beanstalk console override the same settings in configuration files, if they exist. This lets you have default settings in configuration files, and override them with environment-specific settings in the console. For more information about precedence, and other methods of changing settings, see Configuration options.

For details about the various ways you can extend an Elastic Beanstalk Linux-based platform, see the section called "Extending Linux platforms".

Topics

Configuring your Tomcat environment

The Elastic Beanstalk Tomcat platform provides a few platform-specific options in addition to the standard options that all platforms have. These options enable you to configure the Java virtual machine (JVM) that runs on your environment's web servers, and define system properties that provide information configuration strings to your application.

You can use the Elastic Beanstalk console to enable log rotation to Amazon S3 and configure variables that your application can read from the environment.

To configure your Tomcat environment in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.

Container options

You can specify these platform-specific options:

- **Proxy server** – The proxy server to use on your environment instances. By default, nginx is used.

JVM container options

The heap size in the Java virtual machine (JVM) determines how many objects your application can create in memory before *garbage collection* occurs. You can modify the **Initial JVM Heap Size** (`-Xms option`) and a **Maximum JVM Heap Size** (`-Xmx` option). A larger initial heap size allows more objects to be created before garbage collection occurs, but it also means that the garbage collector will take longer to compact the heap. The maximum heap size specifies the maximum amount of memory the JVM can allocate when expanding the heap during heavy activity.

Note

The available memory depends on the Amazon EC2 instance type. For more information about the EC2 instance types available for your Elastic Beanstalk environment, see Instance Types in the *Amazon Elastic Compute Cloud User Guide for Linux Instances*.

The *permanent generation* is a section of the JVM heap that stores class definitions and associated metadata. To modify the size of the permanent generation, type the new size in the **Maximum**

JVM PermGen Size (`-XX:MaxPermSize`) option. This setting applies only to Java 7 and earlier. This option was deprecated in JDK 8 and superseded by the **MaxMetaspace Size** (`-XX:MaxMetaspaceSize`) option.

Important

JDK 17 removed support of the Java `-XX:MaxPermSize` option. Usage of this option with an environment running on an Elastic Beanstalk platform branch with Corretto 17 will result in an error. Elastic Beanstalk released its first platform branch running Tomcat with Corretto 17 on July 13, 2023.
For more information see the following resources.

- Oracle Java documentation website: Removed Java Options
- Oracle Java documentation website: *Class Metadata* section in Other Considerations

For more information about Elastic Beanstalk platforms and their components, see Supported Platforms in the *AWS Elastic Beanstalk Platforms* guide.

Log options

The **Log Options** section has two settings:

- **Instance profile** – Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3** – Specifies whether log files for your application's Amazon EC2 instances are copied to the Amazon S3 bucket associated with your application.

Static files

To improve performance, you can use the **Static files** section to configure the proxy server to serve static files (for example, HTML or images) from a set of directories inside your web application. For each directory, you set the virtual path to directory mapping. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application.

For details about configuring static files using configuration files or the Elastic Beanstalk console, see the section called “Static files”.

Environment properties

In the **Environment Properties** section, you can specify environment configuration settings on the Amazon EC2 instances that are running your application. Environment properties are passed in as key-value pairs to the application.

The Tomcat platform defines a placeholder property named `JDBC_CONNECTION_STRING` for Tomcat environments for passing a connection string to an external database.

Note

If you attach an RDS DB instance to your environment, construct the JDBC connection string dynamically from the Amazon Relational Database Service (Amazon RDS) environment properties provided by Elastic Beanstalk. Use JDBC_CONNECTION_STRING only for database instances that are not provisioned using Elastic Beanstalk.
For more information about using Amazon RDS with your Java application, see Adding an Amazon RDS DB instance to your Java Elastic Beanstalk environment.

For Tomcat platform versions released prior to March 26, 2025, environment variables are accessible using `System.getProperty()`. For example, you could read a property named `API_ENDPOINT` from a variable with the following code.

```
String endpoint = System.getProperty("API_ENDPOINT");
```

Tomcat platform versions released on or after March 26, 2025, can also use `System.getenv` to access plaintext environment variables. You can continue to use `System.getProperty` to access plaintext environment variables. However, environment variables stored as secrets are only available using `System.getenv`. For example, you could read an environment variable named `API_KEY` with the following code.

```
String apiKey = System.getenv("API_KEY");
```

Important

The addition of `System.getenv()` access for environment variables in Tomcat platform versions released on or after March 26, 2025 may cause unexpected behavior in

applications that give environment variables precedence over Java system properties or when explicitly switching from `System.getProperty()` to `System.getenv()`. Since system properties (passed via command line) require shell escaping for special characters while environment variables do not, values may be resolved differently when using environment variables instead of Java system properties.
If your application is affected, consider:

- Removing escape characters from your environment property values when using `System.getenv()`
- Configuring your application to explicitly use `System.getProperty()`
- Testing your application thoroughly when upgrading to ensure consistent behavior

See Environment variables and other software settings for more information.

Tomcat configuration namespaces

You can use a configuration file to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be platform specific or apply to all platforms in the Elastic Beanstalk service as a whole. Configuration options are organized into *namespaces*.

The Tomcat platform supports options in the following namespaces, in addition to the options supported for all Elastic Beanstalk environments:

- `aws:elasticbeanstalk:container:tomcat:jvmoptions` – Modify JVM settings. Options in this namespace correspond to options in the management console, as follows:
  - `Xms` – **JVM command line options**
  - `JVM Options` – **JVM command line options**
- `aws:elasticbeanstalk:environment:proxy` – Choose the environment's proxy server.

The following example configuration file shows the use of the Tomcat-specific configuration options.

Example .ebextensions/tomcat-settings.config

```
option_settings:
  aws:elasticbeanstalk:container:tomcat:jvmoptions:
    Xms: 512m
    JVM Options: '-Xmn128m'
  aws:elasticbeanstalk:application:environment:
    API_ENDPOINT: mywebapi.zkpexsjtmd.us-west-2.elasticbeanstalk.com
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: apache
```

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See Configuration options for more information.

The Amazon Linux AMI (preceding Amazon Linux 2) Tomcat platform

If your Elastic Beanstalk Tomcat environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Notes

- The information in this topic only applies to platform branches based on Amazon Linux AMI (AL1). AL2023/AL2 platform branches are incompatible with previous Amazon Linux AMI (AL1) platform versions and *require different configuration settings*.
- On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information about migrating to a current and fully supported Amazon Linux 2023 platform branch, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

Tomcat configuration namespaces — Amazon Linux AMI (AL1)

The Tomcat Amazon Linux AMI platform supports additional options in the following namespaces:

- `aws:elasticbeanstalk:container:tomcat:jvmoptions` – In addition to the options mentioned earlier on this page for this namespace, older Amazon Linux AMI platform versions also support:
  - `XX:MaxPermSize` – **Maximum JVM permanent generation size**
- `aws:elasticbeanstalk:environment:proxy` – In addition to choosing the proxy server, also configure response compression.

The following example configuration file shows the use of the proxy namespace configuration options.

Example .ebextensions/tomcat-settings.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    GzipCompression: 'true'
    ProxyServer: nginx
```

Include Elastic Beanstalk configurations files — Amazon Linux AMI (AL1)

To deploy `.ebextensions` configuration files, include them in your application source. For a single application, add your `.ebextensions` to a compressed WAR file by running the following command:

Example

```
zip -ur your_application.war .ebextensions
```

For an application requiring multiple WAR files, see Bundling multiple WAR files for Tomcat environments for further instructions.

Bundling multiple WAR files for Tomcat environments

If your web app comprises multiple web application components, you can simplify deployments and reduce operating costs by running components in a single environment, instead of running a separate environment for each component. This strategy is effective for lightweight applications that don't require a lot of resources, and for development and test environments.

To deploy multiple web applications to your environment, combine each component's web application archive (WAR) files into a single source bundle.

To create an application source bundle that contains multiple WAR files, organize the WAR files using the following structure.

```
MyApplication.zip
### .ebextensions
### .platform
### foo.war
### bar.war
### ROOT.war
```

When you deploy a source bundle containing multiple WAR files to an AWS Elastic Beanstalk environment, each application is accessible from a different path off of the root domain name. The preceding example includes three applications: `foo`, `bar`, and `ROOT`. `ROOT.war` is a special file name that tells Elastic Beanstalk to run that application at the root domain, so that the three applications are available at `http://MyApplication.elasticbeanstalk.com/foo`, `http://MyApplication.elasticbeanstalk.com/bar`, and `http://MyApplication.elasticbeanstalk.com`.

The source bundle can include WAR files, an optional `.ebextensions` folder, and an optional `.platform` folder. For details about these optional configuration folders, see the section called "Extending Linux platforms".

To launch an environment (console)

1. Open the Elastic Beanstalk console with this preconfigured link: console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application, or the Docker platform for container-based applications.
3. For **Application code**, choose **Upload your code**.
4. Choose **Local file**, choose **Choose file**, and then open the source bundle.
5. Choose **Review and launch**.
6. Review the available settings, and then choose **Create app**.

For information about creating source bundles, see Create an Elastic Beanstalk application source bundle.

Structuring your project folder

To work when deployed to a Tomcat server, compiled Java Platform Enterprise Edition (*Java EE*) web application archives (WAR files) must be structured according to certain guidelines. Your project directory doesn't have to meet the same standards, but it's a good idea to structure it in the same way to simplify compiling and packaging. Structuring your project folder like the WAR file contents also helps you understand how files are related and how they behave on a web server.

In the following recommended hierarchy, the source code for the web application is placed in a `src` directory, to isolate it from the build script and the WAR file it generates.

```
~/workspace/my-app/
|-- build.sh            - Build script that compiles classes and creates a WAR
|-- README.MD           - Readme file with information about your project, notes
|-- ROOT.war            - Source bundle artifact created by build.sh
`-- src                 - Source code folder
    |-- WEB-INF         - Folder for private supporting files
    |   |-- classes     - Compiled classes
    |   |-- lib         - JAR libraries
    |   |-- tags        - Tag files
    |   |-- tlds        - Tag Library Descriptor files
    |   `-- web.xml     - Deployment Descriptor
    |-- com             - Uncompiled classes
    |-- css             - Style sheets
    |-- images          - Image files
    |-- js              - JavaScript files
    `-- default.jsp     - JSP (JavaServer Pages) webpage
```

The `src` folder contents match what you will package and deploy to the server, with the exception of the `com` folder. The `com` folder contains your uncompiled classes (`.java` files). These need to be compiled and placed in the `WEB-INF/classes` directory to be accessible from your application code.

The `WEB-INF` directory contains code and configurations that are not served publicly on the web server. The other folders at the root of the source directory (`css`, `images`, and `js`) are publicly available at the corresponding path on the web server.

The following example is identical to the preceding project directory, except that it contains more files and subdirectories. This example project includes simple tags, model and support classes, and a Java Server Pages (JSP) file for a `record` resource. It also includes a style sheet and JavaScript for Bootstrap, a default JSP file, and an error page for 404 errors.

`WEB-INF/lib` includes a Java Archive (JAR) file containing the Java Database Connectivity (JDBC) driver for PostgreSQL. `WEB-INF/classes` is empty because class files have not been compiled yet.

```
~/workspace/my-app/
|-- build.sh
|-- README.MD
|-- ROOT.war
```

```
`-- src
    |-- WEB-INF
    |   |-- classes
    |   |-- lib
    |   |   `-- postgresql-9.4-1201.jdbc4.jar
    |   |-- tags
    |   |   `-- header.tag
    |   |-- tlds
    |   |   `-- records.tld
    |   `-- web.xml
    |-- com
    |   `-- myapp
    |       |-- model
    |       |   `-- Record.java
    |       `-- web
    |           `-- ListRecords.java
    |-- css
    |   |-- bootstrap.min.css
    |   `-- myapp.css
    |-- images
    |   `-- myapp.png
    |-- js
    |   `-- bootstrap.min.js
    |-- 404.jsp
    |-- default.jsp
    `-- records.jsp
```

Building a WAR file with a shell script

`build.sh` is a very simple shell script that compiles Java classes, constructs a WAR file, and copies it to the Tomcat `webapps` directory for local testing.

```
cd src
javac -d WEB-INF/classes com/myapp/model/Record.java
javac -classpath WEB-INF/lib/*:WEB-INF/classes -d WEB-INF/classes com/myapp/model/
Record.java
javac -classpath WEB-INF/lib/*:WEB-INF/classes -d WEB-INF/classes com/myapp/web/
ListRecords.java

jar -cvf ROOT.war *.jsp images css js WEB-INF
cp ROOT.war /Library/Tomcat/webapps
mv ROOT.war ../
```

Inside the WAR file, you find the same structure that exists in the `src` directory in the preceding example, excluding the `src/com` folder. The `jar` command automatically creates the `META-INF/MANIFEST.MF` file.

```
~/workspace/my-app/ROOT.war
|-- META-INF
|   `-- MANIFEST.MF
|-- WEB-INF
|   |-- classes
|   |   `-- com
|   |       `-- myapp
|   |           |-- model
|   |           |   `-- Records.class
|   |           `-- web
|   |               `-- ListRecords.class
|   |-- lib
|   |   `-- postgresql-9.4-1201.jdbc4.jar
|   |-- tags
|   |   `-- header.tag
|   |-- tlds
|   |   `-- records.tld
|   `-- web.xml
|-- css
|   |-- bootstrap.min.css
|   `-- myapp.css
|-- images
|   `-- myapp.png
|-- js
|   `-- bootstrap.min.js
|-- 404.jsp
|-- default.jsp
`-- records.jsp
```

Using `.gitignore`

To avoid committing compiled class files and WAR files to your Git repository, or seeing messages about them appear when you run Git commands, add the relevant file types to a file named `.gitignore` in your project folder.

~/workspace/myapp/.gitignore

```
*.zip
```

```
*.class
```

Configuring the proxy server

The Tomcat platform uses nginx (the default) or Apache HTTP Server as the reverse proxy to relay requests from port 80 on the instance to your Tomcat web container listening on port 8080. Elastic Beanstalk provides a default proxy configuration that you can extend or override completely with your own configuration.

Configuring the proxy server on your platform version

All AL2023/AL2 platforms support a uniform proxy configuration feature. For more information about configuring the proxy server on your platform versions running AL2023/AL2, see Reverse proxy configuration.

Configuring the proxy on the Amazon Linux AMI (preceding Amazon Linux 2) Tomcat platform

If your Elastic Beanstalk Tomcat environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Notes

- The information in this topic only applies to platform branches based on Amazon Linux AMI (AL1). AL2023/AL2 platform branches are incompatible with previous Amazon Linux AMI (AL1) platform versions and *require different configuration settings*.
- On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information about migrating to a current and fully supported Amazon Linux 2023 platform branch, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

Choosing a proxy server for your Tomcat environment — Amazon Linux AMI (AL1)

Tomcat platform versions based on Amazon Linux AMI (preceding Amazon Linux 2) use Apache 2.4 for the proxy by default. You can choose to use Apache 2.2 or nginx by including a configuration file in your source code. The following example configures Elastic Beanstalk to use nginx.

Example .ebextensions/nginx-proxy.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: nginx
```

Migrating from Apache 2.2 to Apache 2.4 — Amazon Linux AMI (AL1)

If your application was developed for Apache 2.2, read this section to learn about migrating to Apache 2.4.

Starting with Tomcat platform version 3.0.0 configurations, which were released with the Java with Tomcat platform update on May 24, 2018, Apache 2.4 is the default proxy of the Tomcat platform. The Apache 2.4 `.conf` files are mostly, but not entirely, backward compatible with those of Apache 2.2. Elastic Beanstalk includes default `.conf` files that work correctly with each Apache version. If your application doesn't customize Apache's configuration, as explained in Extending and overriding the default Apache configuration — Amazon Linux AMI (AL1), it should migrate to Apache 2.4 without any issues.

If your application extends or overrides Apache's configuration, you might have to make some changes to migrate to Apache 2.4. For more information, see Upgrading to 2.4 from 2.2 on *The Apache Software Foundation*'s site. As a temporary measure, until you successfully migrate to Apache 2.4, you can choose to use Apache 2.2 with your application by including the following configuration file in your source code.

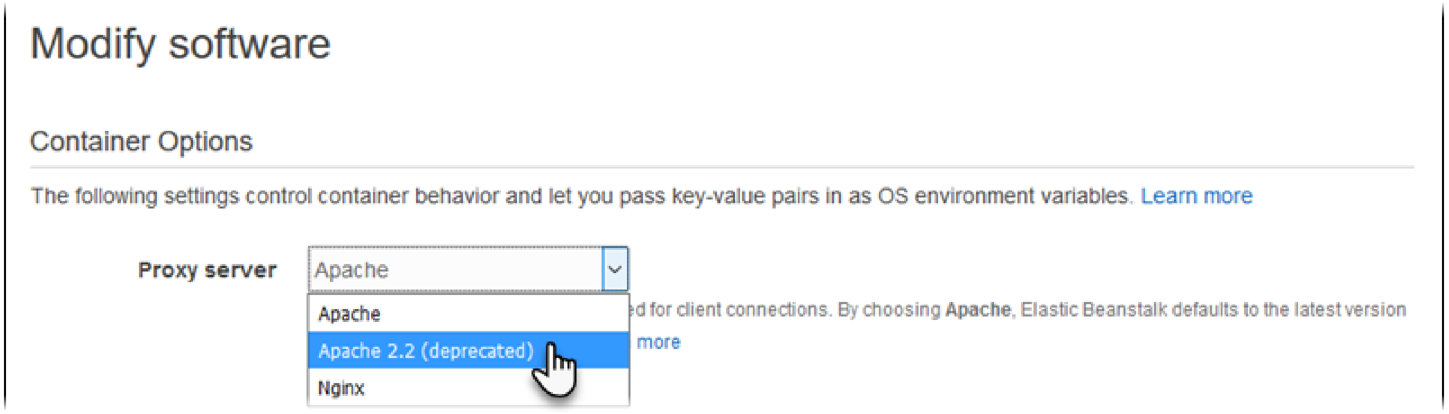
Example .ebextensions/apache-legacy-proxy.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: apache/2.2
```

For a quick fix, you can also select the proxy server in the Elastic Beanstalk console.

To select the proxy in your Tomcat environment in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. For **Proxy server**, choose `Apache 2.2 (deprecated)`.
6. To save the changes choose **Apply** at the bottom of the page.



Modify software

Container Options

The following settings control container behavior and let you pass key-value pairs in as OS environment variables. Learn more

Proxy server: Apache

Apache
Apache 2.2 (deprecated)
Nginx
d for client connections. By choosing Apache, Elastic Beanstalk defaults to the latest version
more

Extending and overriding the default Apache configuration — Amazon Linux AMI (AL1)

You can extend the Elastic Beanstalk default Apache configuration with your additional configuration files. Alternatively, you can override the Elastic Beanstalk default Apache configuration completely.

Note

- All Amazon Linux 2 platforms support a uniform proxy configuration feature. For details about configuring the proxy server on Tomcat platform versions running Amazon Linux 2, see Reverse proxy configuration.
- If you're migrating your Elastic Beanstalk application to an Amazon Linux 2 platform, be sure to also read the information in the section called "Migrate to AL2023/AL2".

To extend the Elastic Beanstalk default Apache configuration, add `.conf` configuration files to a folder named `.ebextensions/httpd/conf.d` in your application source bundle. The Elastic Beanstalk Apache configuration includes `.conf` files in this folder automatically.

```
~/workspace/my-app/
|-- .ebextensions
|   -- httpd
|      -- conf.d
|         -- myconf.conf
|         -- ssl.conf
-- index.jsp
```

For example, the following Apache 2.4 configuration adds a listener on port 5000.

Example .ebextensions/httpd/conf.d/port5000.conf

```
listen 5000
<VirtualHost *:5000>
  <Proxy *>
    Require all granted
  </Proxy>
  ProxyPass / http://localhost:8080/ retry=0
  ProxyPassReverse / http://localhost:8080/
  ProxyPreserveHost on

  ErrorLog /var/log/httpd/elasticbeanstalk-error_log
</VirtualHost>
```

To override the Elastic Beanstalk default Apache configuration completely, include a configuration in your source bundle at `.ebextensions/httpd/conf/httpd.conf`.

```
~/workspace/my-app/
|-- .ebextensions
|   `-- httpd
|       `-- conf
|           `-- httpd.conf
`-- index.jsp
```

If you override the Elastic Beanstalk Apache configuration, add the following lines to your `httpd.conf` to pull in the Elastic Beanstalk configurations for Enhanced health reporting and monitoring in Elastic Beanstalk, response compression, and static files.

```
IncludeOptional conf.d/*.conf
IncludeOptional conf.d/elasticbeanstalk/*.conf
```

If your environment uses Apache 2.2 as its proxy, replace the `IncludeOptional` directives with `Include`. For details about the behavior of these two directives in the two Apache versions, see Include in Apache 2.4, IncludeOptional in Apache 2.4, and Include in Apache 2.2.

Note

To override the default listener on port 80, include a file named `00_application.conf` at `.ebextensions/httpd/conf.d/elasticbeanstalk/` to overwrite the Elastic Beanstalk configuration.

For a working example, take a look at the Elastic Beanstalk default configuration file at `/etc/httpd/conf/httpd.conf` on an instance in your environment. All files in the `.ebextensions/httpd` folder in your source bundle are copied to `/etc/httpd` during deployments.

Extending the default nginx configuration — Amazon Linux AMI (AL1)

To extend Elastic Beanstalk's default nginx configuration, add `.conf` configuration files to a folder named `.ebextensions/nginx/conf.d/` in your application source bundle. The Elastic Beanstalk nginx configuration includes `.conf` files in this folder automatically.

```
~/workspace/my-app/
|-- .ebextensions
|   `-- nginx
|       `-- conf.d
|           |-- elasticbeanstalk
|           |   `-- my-server-conf.conf
|           `-- my-http-conf.conf
`-- index.jsp
```

Files with the .conf extension in the `conf.d` folder are included in the `http` block of the default configuration. Files in the `conf.d/elasticbeanstalk` folder are included in the `server` block within the `http` block.

To override the Elastic Beanstalk default nginx configuration completely, include a configuration in your source bundle at `.ebextensions/nginx/nginx.conf`.

```
~/workspace/my-app/
|-- .ebextensions
|   `-- nginx
|       `-- nginx.conf
`-- index.jsp
```

Notes

- If you override the Elastic Beanstalk nginx configuration, add the following line to your configuration's `server` block to pull in the Elastic Beanstalk configurations for the port 80 listener, response compression, and static files.

  ```
  include conf.d/elasticbeanstalk/*.conf;
  ```

- To override the default listener on port 80, include a file named `00_application.conf` at `.ebextensions/nginx/conf.d/elasticbeanstalk/` to overwrite the Elastic Beanstalk configuration.
- Also include the following line in your configuration's `http` block to pull in the Elastic Beanstalk configurations for Enhanced health reporting and monitoring in Elastic Beanstalk and logging.

```
    include       conf.d/*.conf;
```

For a working example, take a look at the Elastic Beanstalk default configuration file at `/etc/nginx/nginx.conf` on an instance in your environment. All files in the `.ebextensions/nginx` folder in your source bundle are copied to `/etc/nginx` during deployments.

Using the Elastic Beanstalk Java SE platform

This topic describes how to configure, build, and run your Java applications that run on the AWS Elastic Beanstalk Java SE platform.

The Elastic Beanstalk Java SE platform is a set of platform versions for Java web applications that can run on their own from a compiled JAR file. You can compile your application locally or upload the source code with a build script to compile it on-instance. Java SE platform versions are grouped into platform branches, each of which corresponds to a major version of Java.

Note

Elastic Beanstalk doesn't parse your application's JAR file. Keep files that Elastic Beanstalk needs outside of the JAR file. For example, include the `cron.yaml` file of a worker environment at the root of your application's source bundle, next to the JAR file.

Configuration options are available in the Elastic Beanstalk console for modifying the configuration of a running environment. To avoid losing your environment's configuration when you terminate it, you can use saved configurations to save your settings and later apply them to another environment.

To save settings in your source code, you can include configuration files. Settings in configuration files are applied every time you create an environment or deploy your application. You can also use configuration files to install packages, run scripts, and perform other instance customization operations during deployments.

The Elastic Beanstalk Java SE platform includes an nginx server that acts as a reverse proxy, serving cached static content and passing requests to your application. The platform provides configuration options to configure the proxy server to serve static assets from a folder in your source code to reduce the load on your application. For advanced scenarios, you can include your own .conf files in your source bundle to extend Elastic Beanstalk's proxy configuration or overwrite it completely.

If you only provide a single JAR file for your application source (on its own, not within a source bundle), Elastic Beanstalk renames your JAR file to `application.jar`, and then runs it using `java -jar application.jar`. To configure the processes that run on the server instances in your environment, include an optional Procfile in your source bundle. A `Procfile` is required if you have more than one JAR in your source bundle root, or if you want to customize the java command to set JVM options.

We recommend that you always provide a `Procfile` in the source bundle alongside your application. This way you precisely control which processes Elastic Beanstalk runs for your application and which arguments these processes receive.

To compile Java classes and run other build commands on the EC2 instances in your environment at deploy time, include a Buildfile in your application source bundle. A `Buildfile` lets you deploy your source code as-is and build on the server instead of compiling JARs locally. The Java SE platform includes common build tools to let you build on-server.

For details about the various ways you can extend an Elastic Beanstalk Linux-based platform, see the section called "Extending Linux platforms".

Configuring your Java SE environment

The Java SE platform settings let you fine-tune the behavior of your Amazon EC2 instances. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration using the Elastic Beanstalk console.

Use the Elastic Beanstalk console to enable log rotation to Amazon S3 and configure variables that your application can read from the environment.

To configure your Java SE environment in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.

Log options

The Log Options section has two settings:

- **Instance profile** – Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3** – Specifies whether log files for your application's Amazon EC2 instances are copied to the Amazon S3 bucket associated with your application.

Static files

To improve performance, you can use the **Static files** section to configure the proxy server to serve static files (for example, HTML or images) from a set of directories inside your web application. For each directory, you set the virtual path to directory mapping. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application.

For details about configuring static files using configuration files or the Elastic Beanstalk console, see the section called "Static files".

Environment properties

The **Environment Properties** section lets you specify environment configuration settings on the Amazon EC2 instances that are running your application. Environment properties are passed in as key-value pairs to the application.

Inside the Java SE environment running in Elastic Beanstalk, environment variables are accessible using the `System.getenv()`. For example, you could read a property named `API_ENDPOINT` to a variable with the following code:

```
String endpoint = System.getenv("API_ENDPOINT");
```

See Environment variables and other software settings for more information.

Java SE configuration namespace

You can use a configuration file to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be platform specific or apply to all platforms in the Elastic Beanstalk service as a whole. Configuration options are organized into *namespaces*.

The Java SE platform doesn't define any platform-specific namespaces. You can configure the proxy to serve static files by using the `aws:elasticbeanstalk:environment:proxy:staticfiles` namespace. For details and an example, see the section called “Static files”.

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See Configuration options for more information.

The Amazon Linux AMI (preceding Amazon Linux 2) Java SE platform

If your Elastic Beanstalk Java SE environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Notes

- The information in this topic only applies to platform branches based on Amazon Linux AMI (AL1). AL2023/AL2 platform branches are incompatible with previous Amazon Linux AMI (AL1) platform versions and *require different configuration settings*.
- On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information about migrating to a current and fully supported Amazon Linux 2023 platform branch, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

Java SE configuration namespaces — Amazon Linux AMI (AL1)

You can use a configuration file to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be platform specific or apply to all platforms in the Elastic Beanstalk service as a whole. Configuration options are organized into *namespaces*.

The Java SE platform supports one platform-specific configuration namespace in addition to the namespaces supported by all platforms. The `aws:elasticbeanstalk:container:java:staticfiles` namespace lets you define options that map paths on your web application to folders in your application source bundle that contain static content.

For example, this option_settings snippet defines two options in the static files namespace. The first maps the path `/public` to a folder named `public`, and the second maps the path `/images` to a folder named `img`:

```
option_settings:
  aws:elasticbeanstalk:container:java:staticfiles:
    /html: statichtml
    /images: staticimages
```

The folders that you map using this namespace must be actual folders in the root of your source bundle. You cannot map a path to a folder in a JAR file.

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See Configuration options for more information.

Building JARs on-server with a Buildfile

You can build your application's class files and JAR(s) on the EC2 instances in your environment by invoking a build command from a `Buildfile` file in your source bundle.

Commands in a `Buildfile` are only run once and must terminate upon completion, whereas commands in a Procfile are expected to run for the life of the application and will be restarted if they terminate. To run the JARs in your application, use a `Procfile`.

For details about the placement and syntax of a `Buildfile`, see Buildfile and Procfile.

The following `Buildfile` example runs Apache Maven to build a web application from source code. For a sample application that uses this feature, see Java web application samples.

Example Buildfile

```
build: mvn assembly:assembly -DdescriptorId=jar-with-dependencies
```

The Java SE platform includes the following build tools, which you can invoke from your build script:

- `javac` – Java compiler
- `ant` – Apache Ant
- `mvn` – Apache Maven
- `gradle` – Gradle

Configuring the application process with a Procfile

If you have more than one JAR file in the root of your application source bundle, you must include a `Procfile` file that tells Elastic Beanstalk which JAR(s) to run. You can also include a `Procfile` file for a single JAR application to configure the Java virtual machine (JVM) that runs your application.

We recommend that you always provide a `Procfile` in the source bundle alongside your application. This way you precisely control which processes Elastic Beanstalk runs for your application and which arguments these processes receive.

For details about writing and using a `Procfile` see Buildfile and Procfile.

Example Procfile

```
web: java -Xms256m -jar server.jar
cache: java -jar mycache.jar
web_foo: java -jar other.jar
```

The command that runs the main JAR in your application must be called `web`, and it must be the first command listed in your `Procfile`. The nginx server forwards all HTTP requests that it receives from your environment's load balancer to this application.

Elastic Beanstalk assumes that all entries in the Procfile should run at all times and automatically restarts any application defined in the Procfile that terminates. To run commands that will terminate and should not be restarted, use a `Buildfile`.

Using a Procfile on Amazon Linux AMI (preceding Amazon Linux 2)

If your Elastic Beanstalk Java SE environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Notes

- The information in this topic only applies to platform branches based on Amazon Linux AMI (AL1). AL2023/AL2 platform branches are incompatible with previous Amazon Linux AMI (AL1) platform versions and *require different configuration settings*.
- On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information about migrating to a current and fully supported Amazon Linux 2023 platform branch, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

Port passing — Amazon Linux AMI (AL1)

By default, Elastic Beanstalk configures the nginx proxy to forward requests to your application on port 5000. You can override the default port by setting the `PORT` environment property to the port on which your main application listens.

If you use a `Procfile` to run multiple applications, Elastic Beanstalk on Amazon Linux AMI platform versions expects each additional application to listen on a port 100 higher than the previous one. Elastic Beanstalk sets the PORT variable accessible from within each application to the port that it expects the application to run on. You can access this variable within your application code by calling `System.getenv("PORT")`.

In the preceding `Procfile` example, the `web` application listens on port 5000, `cache` listens on port 5100, and `web_foo` listens on port 5200. `web` configures its listening port by reading the `PORT` variable, and adds 100 to that number to determine which port `cache` is listening on so that it can send it requests.

Configuring the proxy server

Elastic Beanstalk uses nginx as the reverse proxy to map your application to your Elastic Load Balancing load balancer on port 80. Elastic Beanstalk provides a default nginx configuration that you can either extend or override completely with your own configuration.

By default, Elastic Beanstalk configures the nginx proxy to forward requests to your application on port 5000. You can override the default port by setting the PORT environment property to the port on which your main application listens.

Note

The port that your application listens on doesn't affect the port that the nginx server listens to receive requests from the load balancer.

Configuring the proxy server on your platform version

All AL2023/AL2 platforms support a uniform proxy configuration feature. For more information about configuring the proxy server on your platform versions running AL2023/AL2, see Reverse proxy configuration.

Configuring the proxy on Amazon Linux AMI (preceding Amazon Linux 2)

If your Elastic Beanstalk Java SE environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Notes

- The information in this topic only applies to platform branches based on Amazon Linux AMI (AL1). AL2023/AL2 platform branches are incompatible with previous Amazon Linux AMI (AL1) platform versions and *require different configuration settings*.
- On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information about migrating to a current and fully supported Amazon Linux 2023 platform branch, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

Extending and overriding the default proxy configuration — Amazon Linux AMI (AL1)

To extend Elastic Beanstalk's default nginx configuration, add `.conf` configuration files to a folder named `.ebextensions/nginx/conf.d/` in your application source bundle. Elastic Beanstalk's nginx configuration includes `.conf` files in this folder automatically.

```
~/workspace/my-app/
|-- .ebextensions
|   `-- nginx
|       `-- conf.d
|           `-- myconf.conf
`-- web.jar
```

To override Elastic Beanstalk's default nginx configuration completely, include a configuration in your source bundle at `.ebextensions/nginx/nginx.conf`:

```
~/workspace/my-app/
|-- .ebextensions
|   `-- nginx
|       `-- nginx.conf
`-- web.jar
```

If you override Elastic Beanstalk's nginx configuration, add the following line to your `nginx.conf` to pull in Elastic Beanstalk's configurations for Enhanced health reporting and monitoring in Elastic Beanstalk, automatic application mappings, and static files.

```
 include conf.d/elasticbeanstalk/*.conf;
```

The following example configuration from the Scorekeep sample application overrides Elastic Beanstalk's default configuration to serve a static web application from the `public` subdirectory of `/var/app/current`, where the Java SE platform copies the application source code. The `/api` location forwards traffic to routes under `/api/` to the Spring application listening on port 5000. All other traffic is served by the web app at the root path.

Example

```
user                    nginx;
error_log               /var/log/nginx/error.log warn;
pid                     /var/run/nginx.pid;
worker_processes        auto;
worker_rlimit_nofile    33282;
```

```
events {
    worker_connections  1024;
}

http {
  include       /etc/nginx/mime.types;
  default_type  application/octet-stream;

  log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                    '$status $body_bytes_sent "$http_referer" '
                    '"$http_user_agent" "$http_x_forwarded_for"';

  include       conf.d/*.conf;

  map $http_upgrade $connection_upgrade {
      default     "upgrade";
  }

  server {
      listen        80 default_server;
      root /var/app/current/public;

      location / {
      }git pull


      location /api {
          proxy_pass          http://127.0.0.1:5000;
          proxy_http_version  1.1;

          proxy_set_header    Connection          $connection_upgrade;
          proxy_set_header    Upgrade             $http_upgrade;
          proxy_set_header    Host                $host;
          proxy_set_header    X-Real-IP           $remote_addr;
          proxy_set_header    X-Forwarded-For     $proxy_add_x_forwarded_for;
      }

      access_log    /var/log/nginx/access.log main;

      client_header_timeout 60;
      client_body_timeout   60;
      keepalive_timeout     60;
      gzip                  off;
      gzip_comp_level       4;

      # Include the Elastic Beanstalk generated locations
      include conf.d/elasticbeanstalk/01_static.conf;
      include conf.d/elasticbeanstalk/healthd.conf;
  }
}
```

Adding an Amazon RDS DB instance to your Java Elastic Beanstalk environment

This topic provides instructions to create an Amazon RDS using the Elastic Beanstalk console. You can use an Amazon Relational Database Service (Amazon RDS) DB instance to store data that your application gathers and modifies. The database can be attached to your environment and managed by Elastic Beanstalk, or created and managed externally.

If you are using Amazon RDS for the first time, add a DB instance to a test environment by using the Elastic Beanstalk console and verify that your application can connect to it.

To add a DB instance to your environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Choose a DB engine, and enter a user name and password.
6. To save the changes choose **Apply** at the bottom of the page.

Adding a DB instance takes about 10 minutes. When the environment update is complete, the DB instance's hostname and other connection information are available to your application through the following environment properties:

Property name	Description	Property value
`RDS_HOSTNAME`	The hostname of the DB instance.	On the **Connectivity & security** tab on the Amazon RDS console: **Endpoint**.
`RDS_PORT`	The port where the DB instance accepts connectio ns. The default value varies among DB engines.	On the **Connectivity & security** tab on the Amazon RDS console: **Port**.
`RDS_DB_NAME`	The database name, **ebdb**.	On the **Configuration** tab on the Amazon RDS console: **DB Name**.
`RDS_USERNAME`	The username that you configured for your database.	On the **Configuration** tab on the Amazon RDS console: **Master username**.
`RDS_PASSWORD`	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

For more information about configuring an internal DB instance, see Adding a database to your Elastic Beanstalk environment. For instructions on configuring an external database for use with Elastic Beanstalk, see Using Elastic Beanstalk with Amazon RDS.

To connect to the database, add the appropriate driver JAR file to your application, load the driver class in your code, and create a connection object with the environment properties provided by Elastic Beanstalk.

Sections

Downloading the JDBC driver

You will need the JAR file of the JDBC driver for the DB engine that you choose. Save the JAR file in your source code and include it in your classpath when you compile the class that creates connections to the database.

You can find the latest driver for your DB engine in the following locations:

- **MySQL** – MySQL Connector/J
- **Oracle SE-1** – Oracle JDBC Driver
- **Postgres** – PostgreSQL JDBC Driver
- **SQL Server** – Microsoft JDBC Driver

To use the JDBC driver, call `Class.forName()` to load it before creating the connection with `DriverManager.getConnection()` in your code.

JDBC uses a connection string in the following format:

```
jdbc:driver://hostname:port/dbName?user=userName&password=password
```

You can retrieve the hostname, port, database name, user name, and password from the environment variables that Elastic Beanstalk provides to your application. The driver name is specific to your database type and driver version. The following are example driver names:

- `mysql` for MySQL
- `postgresql` for PostgreSQL
- `oracle:thin` for Oracle Thin
- `oracle:oci` for Oracle OCI
- `oracle:oci8` for Oracle OCI 8
- `oracle:kprb` for Oracle KPRB
- `sqlserver` for SQL Server

Connecting to a database (Java SE platforms)

In a Java SE environment, use `System.getenv()` to read the connection variables from the environment. The following example code shows a class that creates a connection to a PostgreSQL database.

```
private static Connection getRemoteConnection() {
    if (System.getenv("RDS_HOSTNAME") != null) {
      try {
      Class.forName("org.postgresql.Driver");
      String dbName = System.getenv("RDS_DB_NAME");
      String userName = System.getenv("RDS_USERNAME");
      String password = System.getenv("RDS_PASSWORD");
      String hostname = System.getenv("RDS_HOSTNAME");
      String port = System.getenv("RDS_PORT");
      String jdbcUrl = "jdbc:postgresql://" + hostname + ":" + port + "/" + dbName + "?
user=" + userName + "&password=" + password;
      logger.trace("Getting remote connection with connection string from environment
 variables.");
      Connection con = DriverManager.getConnection(jdbcUrl);
      logger.info("Remote connection successful.");
      return con;
    }
    catch (ClassNotFoundException e) { logger.warn(e.toString());}
    catch (SQLException e) { logger.warn(e.toString());}
    }
    return null;
  }
```

Connecting to a database (Tomcat platforms)

In a Tomcat environment, environment properties are provided as system properties that are accessible with `System.getProperty()`.

The following example code shows a class that creates a connection to a PostgreSQL database.

```
private static Connection getRemoteConnection() {
    if (System.getProperty("RDS_HOSTNAME") != null) {
      try {
      Class.forName("org.postgresql.Driver");
      String dbName = System.getProperty("RDS_DB_NAME");
      String userName = System.getProperty("RDS_USERNAME");
      String password = System.getProperty("RDS_PASSWORD");
      String hostname = System.getProperty("RDS_HOSTNAME");
      String port = System.getProperty("RDS_PORT");
      String jdbcUrl = "jdbc:postgresql://" + hostname + ":" + port + "/" + dbName + "?
user=" + userName + "&password=" + password;
      logger.trace("Getting remote connection with connection string from environment
 variables.");
      Connection con = DriverManager.getConnection(jdbcUrl);
      logger.info("Remote connection successful.");
      return con;
    }
    catch (ClassNotFoundException e) { logger.warn(e.toString());}
    catch (SQLException e) { logger.warn(e.toString());}
    }
    return null;
  }
```

If you have trouble getting a connection or running SQL statements, try placing the following code in a JSP file. This code connects to a DB instance, creates a table, and writes to it.

```
<%@ page import="java.sql.*" %>
<%
  // Read RDS connection information from the environment
  String dbName = System.getProperty("RDS_DB_NAME");
  String userName = System.getProperty("RDS_USERNAME");
  String password = System.getProperty("RDS_PASSWORD");
  String hostname = System.getProperty("RDS_HOSTNAME");
  String port = System.getProperty("RDS_PORT");
  String jdbcUrl = "jdbc:mysql://" + hostname + ":" +
    port + "/" + dbName + "?user=" + userName + "&password=" + password;

  // Load the JDBC driver
  try {
    System.out.println("Loading driver...");
    Class.forName("com.mysql.jdbc.Driver");
    System.out.println("Driver loaded!");
  } catch (ClassNotFoundException e) {
    throw new RuntimeException("Cannot find the driver in the classpath!", e);
  }

  Connection conn = null;
  Statement setupStatement = null;
  Statement readStatement = null;
  ResultSet resultSet = null;
  String results = "";
  int numresults = 0;
  String statement = null;

  try {
    // Create connection to RDS DB instance
    conn = DriverManager.getConnection(jdbcUrl);

    // Create a table and write two rows
    setupStatement = conn.createStatement();
    String createTable = "CREATE TABLE Beanstalk (Resource char(50));";
    String insertRow1 = "INSERT INTO Beanstalk (Resource) VALUES ('EC2 Instance');";
    String insertRow2 = "INSERT INTO Beanstalk (Resource) VALUES ('RDS Instance');";

    setupStatement.addBatch(createTable);
    setupStatement.addBatch(insertRow1);
    setupStatement.addBatch(insertRow2);
    setupStatement.executeBatch();
    setupStatement.close();

  } catch (SQLException ex) {
    // Handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
  } finally {
    System.out.println("Closing the connection.");
    if (conn != null) try { conn.close(); } catch (SQLException ignore) {}
  }

  try {
    conn = DriverManager.getConnection(jdbcUrl);

    readStatement = conn.createStatement();
    resultSet = readStatement.executeQuery("SELECT Resource FROM Beanstalk;");

    resultSet.first();
    results = resultSet.getString("Resource");
    resultSet.next();
    results += ", " + resultSet.getString("Resource");

    resultSet.close();
    readStatement.close();
    conn.close();

  } catch (SQLException ex) {
    // Handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
  } finally {
      System.out.println("Closing the connection.");
     if (conn != null) try { conn.close(); } catch (SQLException ignore) {}
  }
%>
```

To display the results, place the following code in the body of the HTML portion of the JSP file.

```
<p>Established connection to RDS. Read first two rows: <%= results %></p>
```

Troubleshooting database connections

If you run into issues connecting to a database from within your application, review the web container log and database.

Reviewing logs

You can view all the logs from your Elastic Beanstalk environment from within Eclipse. If you don't have the AWS Explorer view open, choose the arrow next to the orange AWS icon in the toolbar, and then choose **Show AWS Explorer View**. Expand **AWS Elastic Beanstalk** and your environment name, and then open the context (right-click) menu for the server. Choose **Open in WTP Server Editor**.

Choose the **Log** tab of the **Server** view to see the aggregate logs from your environment. To open the latest logs, choose the **Refresh** button at the upper right corner of the page.

Scroll down to locate the Tomcat logs in `/var/log/tomcat7/catalina.out`. If you loaded the webpage from our earlier example several times, you might see the following.

```
-------------------------------------
/var/log/tomcat7/catalina.out
-------------------------------------
INFO: Server startup in 9285 ms
Loading driver...
Driver loaded!
```

```
SQLException: Table 'Beanstalk' already exists
SQLState: 42S01
VendorError: 1050
Closing the connection.
Closing the connection.
```

All information that the web application sends to standard output appears in the web container log. In the previous example, the application tries to create the table every time the page loads. This results in catching a SQL exception on every page load after the first one.

As an example, the preceding is acceptable. But in actual applications, keep your database definitions in schema objects, perform transactions from within model classes, and coordinate requests with controller servlets.

Connecting to an RDS DB Instance

You can connect directly to the RDS DB instance in your Elastic Beanstalk environment by using the MySQL client application.

First, open the security group to your RDS DB instance to allow traffic from your computer.

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Next to **Endpoint**, choose the Amazon RDS console link.
6. On the **RDS Dashboard** instance details page, under **Security and Network**, choose the security group starting with *rds-* next to **Security Groups**.

   Note

   The database might have multiple entries labeled **Security Groups**. Use the first, which starts with *awseb*, only if you have an older account that doesn't have a default Amazon Virtual Private Cloud (Amazon VPC).

7. In **Security group details**, choose the **Inbound** tab, and then choose **Edit**.
8. Add a rule for MySQL (port 3306) that allows traffic from your IP address, specified in CIDR format.

9. Choose **Save**. The changes take effect immediately.

Return to the Elastic Beanstalk configuration details for your environment and note the endpoint. You will use the domain name to connect to the RDS DB instance.

Install the MySQL client and initiate a connection to the database on port 3306. On Windows, install MySQL Workbench from the MySQL home page and follow the prompts.

On Linux, install the MySQL client using the package manager for your distribution. The following example works on Ubuntu and other Debian derivatives.

```
// Install MySQL client
$ sudo apt-get install mysql-client-5.5
...
// Connect to database
$ mysql -h aas839jo2vwhwb.cnubrrfwfka8.us-west-2.rds.amazonaws.com -u username -
ppassword ebdb
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 117
Server version: 5.5.40-log Source distribution
...
```

After you have connected, you can run SQL commands to see the status of the database, whether your tables and rows were created, and other information.

```
mysql> SELECT Resource from Beanstalk;
+--------------+
| Resource     |
+--------------+
| EC2 Instance |
| RDS Instance |
+--------------+
2 rows in set (0.01 sec)
```

Java tools and resources

There are several places you can go to get additional help when developing your Java applications.

Resource	Description
The AWS Java Development Forum	Post your questions and get feedback.
Java Developer Center	One-stop shop for sample code, documentation, tools, and additional resources.

Deploying Node.js applications with Elastic Beanstalk

This chapter provides instructions for configuring and deploying your Node.js web application to AWS Elastic Beanstalk. It also provides walkthroughs for common tasks such as database integration and working with the Express framework. Elastic Beanstalk makes it easy to deploy, manage, and scale your Node.js web applications using Amazon Web Services.

You can deploy your application in just a few minutes using the Elastic Beanstalk Command Line Interface (EB CLI) or by using the Elastic Beanstalk console. After you deploy your Elastic Beanstalk application, you can continue to use the EB CLI to manage your application and environment, or you can use the Elastic Beanstalk console, AWS CLI, or the APIs.

Follow the QuickStart for Node.js for step-by-step instructions to create and deploy a *Hello World* Node.js web application with the EB CLI.

Topics

QuickStart: Deploy a Node.js application to Elastic Beanstalk

This QuickStart tutorial walks you through the process of creating a Node.js application and deploying it to an AWS Elastic Beanstalk environment.

Note

Tutorial examples are intended for demonstration. Do not use the application for production traffic.

Sections

Your AWS account

If you're not already an AWS customer, you need to create an AWS account. Signing up enables you to access Elastic Beanstalk and other AWS services that you need.

If you already have an AWS account, you can move on to Prerequisites.

Create an AWS account

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.
2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.
2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Prerequisites

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

EB CLI

This tutorial uses the Elastic Beanstalk Command Line Interface (EB CLI). For details on installing and configuring the EB CLI, see Install EB CLI with setup script (recommended) and Configure the EB CLI.

Node.js

Install Node.js on your local machine by following How to install Node.js on the Node.js website.

Verify your Node.js installation by running the following command.

```
~$ node -v
```

Step 1: Create a Node.js application

Create a project directory.

```
~$ mkdir eb-nodejs
~$ cd eb-nodejs
```

Next, create an application that you'll deploy using Elastic Beanstalk. We'll create a "Hello World" RESTful web service.

Example ~/eb-nodejs/server.js

```
const http = require('node:http');

const hostname = '127.0.0.1';
const port = 8080;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello Elastic Beanstalk!\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

This application opens a listener on port 8080. Elastic Beanstalk forwards requests to your application on port 8080 by default for Node.js.

Step 2: Run your application locally

Run the following command to run your application locally.

```
~/eb-nodejs$ node server.js
```

You should see the following text.

```
Server running at http://127.0.0.1:8080/
```

Enter the URL address `http://127.0.0.1:8080/` in your web browser. The browser should display “Hello Elastic Beanstalk!”.

Step 3: Deploy your Node.js application with the EB CLI

Run the following commands to create an Elastic Beanstalk environment for this application.

To create an environment and deploy your Node.js application

1. Initialize your EB CLI repository with the **eb init** command.

   ```
   ~/eb-nodejs$ eb init -p node.js nodejs-tutorial --region us-east-2
   ```

   This command creates an application named `nodejs-tutorial` and configures your local repository to create environments with the latest Node.js platform version.

2. (Optional) Run **eb init** again to configure a default key pair so that you can use SSH to connect to the EC2 instance running your application.

   ```
   ~/eb-nodejs$ eb init
   Do you want to set up SSH for your instances?
   (y/n): y
   Select a keypair.
   1) my-keypair
   2) [ Create new KeyPair ]
   ```

   Select a key pair if you have one already, or follow the prompts to create one. If you don't see the prompt or need to change your settings later, run **eb init -i**.

3. Create an environment and deploy your application to it with **eb create**. Elastic Beanstalk automatically builds a zip file for your application and deploys it to an EC2 instance in the environment. After deploying your application, Elastic Beanstalk starts it on port 8080.

   ```
   ~/eb-nodejs$ eb create nodejs-env
   ```

   It takes about five minutes for Elastic Beanstalk to create your environment.

Step 4: Run your application on Elastic Beanstalk

When the process to create your environment completes, open your website with **eb open**.

```
~/eb-nodejs$ eb open
```

Congratulations! You've deployed a Node.js application with Elastic Beanstalk! This opens a browser window using the domain name created for your application.

Step 5: Clean up

You can terminate your environment when you finish working with your application. Elastic Beanstalk terminates all AWS resources associated with your environment.

To terminate your Elastic Beanstalk environment with the EB CLI run the following command.

```
~/eb-nodejs$ eb terminate
```

AWS resources for your application

You just created a single instance application. It serves as a straightforward sample application with a single EC2 instance, so it doesn't require load balancing or auto scaling. For single instance applications Elastic Beanstalk creates the following AWS resources:

- **EC2 instance** – An Amazon EC2 virtual machine configured to run web apps on the platform you choose.

  Each platform runs a different set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination thereof. Most platforms use either Apache or nginx as a reverse proxy that processes web traffic in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group** – An Amazon EC2 security group configured to allow incoming traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic is not allowed on other ports.
- **Amazon S3 bucket –** A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms –** Two CloudWatch alarms that monitor the load on the instances in your environment and are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack –** Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.

- **Domain name –** A domain name that routes to your web app in the form `subdomain.region`*.elasticbeanstalk.com*.

Elastic Beanstalk manages all of these resources. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Next steps

After you have an environment running an application, you can deploy a new version of the application or a different application at any time. Deploying a new application version is very quick because it doesn't require provisioning or restarting EC2 instances. You can also explore your new environment using the Elastic Beanstalk console. For detailed steps, see Explore your environment in the *Getting started* chapter of this guide.

> **Try more tutorials**
>
> If you'd like to try other tutorials with different example applications, see More Elastic Beanstalk example applications and tutorials for Node.js.

After you deploy a sample application or two and are ready to start developing and running Node.js applications locally, see Setting up your Node.js development environment for Elastic Beanstalk.

Deploy with the Elastic Beanstalk console

You can also use the Elastic Beanstalk console to launch the sample application. For detailed steps, see Create an example application in the *Getting started* chapter of this guide.

Setting up your Node.js development environment for Elastic Beanstalk

This topic provides instructions to set up a Node.js development environment to test your application locally prior to deploying it to AWS Elastic Beanstalk. It also references websites that provide installation instructions for useful tools.

Topics

- Install Node.js
- Confirm npm installation
- Install the AWS SDK for Node.js
- Install the Express generator
- Set up an Express framework and server

Install Node.js

Install Node.js to run Node.js applications locally. If you don't have a preference, get the latest version supported by Elastic Beanstalk. See Node.js in the *AWS Elastic Beanstalk Platforms* document for a list of supported versions.

Download Node.js at nodejs.org.

Confirm npm installation

Node.js uses the npm package manager to help you install tools and frameworks for use in your application. Since npm is distributed with Node.js, you will automatically install it when you download and install Node.js. To confirm you have npm installed you can run the following command:

```
$ npm -v
```

For more information on npm, visit the npmjs website.

Install the AWS SDK for Node.js

If you need to manage AWS resources from within your application, install the AWS SDK for JavaScript in Node.js. Install the SDK with npm:

```
$ npm install aws-sdk
```

Visit the AWS SDK for JavaScript in Node.js homepage for more information.

Install the Express generator

Express is a web application framework that runs on Node.js. To use it, first install the Express generator command line application. Once the Express generator is installed, you can run the

express command to generate a base project structure for your web application. Once the base project, files, and dependencies are installed you can start up a local Express server on your development machine.

Note

- These steps walk you through installing the Express generator on a Linux operating system.
- For Linux, depending on your permission level to system directories, you might need to prefix some of these commands with `sudo`.

To install the Express generator on your development environment

1. Create a working directory for your Express framework and server.

   ```
   ~$ mkdir node-express
   ~$ cd node-express
   ```

2. Install Express globally so that you have access to the `express` command.

   ```
   ~/node-express$ npm install -g express-generator
   ```

3. Depending on your operating system, you may need to set your path to run the `express` command. The output from the previous step provides information if you need to set your path variable. The following is an example for Linux.

   ```
   ~/node-express$ export PATH=$PATH:/usr/local/share/npm/bin/express
   ```

   When you follow the tutorials in this chapter, you'll need to run the **express** command from different directories. Each tutorial sets up a base Express project structure in it's own directory.

You have now installed the Express command line generator. You can use it to create a framework directory for your web application, set up dependencies, and start up the web app server. Next, we'll go through the steps to accomplish this in the `node-express` directory that we created.

Set up an Express framework and server

Follow these steps to create the base Express framework directories and contents. The tutorials in this chapter also include these steps to set up the base Express framework in each of the tutorial's application directories.

To set up an Express framework and server

1. Run the `express` command. This generates `package.json`, `app.js`, and a few directories.

   ```
   ~/node-express$ express
   ```

   When prompted, type **y** if you want to continue.
2. Set up local dependencies.

   ```
   ~/node-express$ npm install
   ```
3. Verify the web app server starts up.

   ```
   ~/node-express$ npm start
   ```

   You should see output similar to the following:

   ```
   > nodejs@0.0.0 start /home/local/user/node-express
   > node ./bin/www
   ```

   The server runs on port 3000 by default. To test it, run `curl http://localhost:3000` in another terminal, or open a browser on the local computer and enter URL address `http://localhost:3000`.

   Press **Ctrl+C** to stop the server.

Using the Elastic Beanstalk Node.js platform

This topic describes how to configure, build, and run your Node.js applications on Elastic Beanstalk.

AWS Elastic Beanstalk supports a number of platform branches for different versions of the Node.js programming language. See Node.js in the *AWS Elastic Beanstalk Platforms* document for a full list.

Elastic Beanstalk provides configuration options that you can use to customize the software that runs on the EC2 instances in your Elastic Beanstalk environment. You can configure environment variables required by your application, enable log rotation to Amazon S3, and map folders in your application source that contain static files to paths served by the proxy server.

Configuration options are available in the Elastic Beanstalk console for modifying the configuration of a running environment. To avoid losing your environment's configuration when you terminate it, you can use saved configurations to save your settings and later apply them to another environment.

To save settings in your source code, you can include configuration files. Settings in configuration files are applied every time you create an environment or deploy your application. You can also use configuration files to install packages, run scripts, and perform other instance customization operations during deployments.

You can include a `Package.json` file in your source bundle to install packages during deployment, to provide a start command, and to specify the Node.js version that you want your application to use. You can include an `npm-shrinkwrap.json` file to lock down dependency versions.

The Node.js platform includes a proxy server to serve static assets, forward traffic to your application, and compress responses. You can extend or override the default proxy configuration for advanced scenarios.

There are several options to start your application. You can add a Procfile to your source bundle to specify the command that starts your application. If you don't provide a `Procfile` but provide a `package.json` file, Elastic Beanstalk runs `npm start`. If you don't provide that either, Elastic Beanstalk looks for the `app.js` or `server.js` file, in this order, and runs the script.

Settings applied in the Elastic Beanstalk console override the same settings in configuration files, if they exist. This lets you have default settings in configuration files, and override them with environment-specific settings in the console. For more information about precedence, and other methods of changing settings, see Configuration options.

For details about the various ways you can extend an Elastic Beanstalk Linux-based platform, see the section called “Extending Linux platforms”.

Configuring your Node.js environment

You can use the Node.js platform settings to fine-tune the behavior of your Amazon EC2 instances. You can edit the Amazon EC2 instance configuration for your Elastic Beanstalk environment using the Elastic Beanstalk console.

Use the Elastic Beanstalk console to enable log rotation to Amazon S3 and configure variables that your application can read from the environment.

To configure your Node.js environment in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.

Container options

You can specify these platform-specific options:

- **Proxy server –** The proxy server to use on your environment instances. By default, NGINX is used.

Log options

The **Log Options** section has two settings:

- **Instance profile –** Specifies the instance profile that has permission to access the Amazon S3 bucket that's associated with your application.
- **Enable log file rotation to Amazon S3 –** Specifies whether log files for your application's Amazon EC2 instances are copied to the Amazon S3 bucket associated with your application.

Static files

To improve performance, you can use the **Static files** section to configure the proxy server to serve static files (for example, HTML or images) from a set of directories inside your web application. For each directory, you set the virtual path to directory mapping. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application.

For details about configuring static files using configuration files or the Elastic Beanstalk console, see the section called "Static files".

Environment properties

Use the **Environment Properties** section to specify environment configuration settings on the Amazon EC2 instances that are running your application. These settings are passed in as key-value pairs to the application.

Inside the Node.js environment that runs in AWS Elastic Beanstalk, you can access the environment variables by running `process.env.ENV_VARIABLE`.

```
var endpoint = process.env.API_ENDPOINT
```

The Node.js platform sets the PORT environment variable to the port that the proxy server passes traffic to. For more information, see Configuring the proxy server.

See Environment variables and other software settings for more information.

Configuring an Amazon Linux AMI (preceding Amazon Linux 2) Node.js environment

The following console software configuration categories are supported only on an Elastic Beanstalk Node.js environment that uses an Amazon Linux AMI platform version (preceding Amazon Linux 2).

Notes

- The information in this topic only applies to platform branches based on Amazon Linux AMI (AL1). AL2023/AL2 platform branches are incompatible with previous Amazon Linux AMI (AL1) platform versions and *require different configuration settings*.
- On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information about migrating to a current and fully supported Amazon Linux 2023 platform branch, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

Container options — Amazon Linux AMI (AL1)

On the configuration page, specify the following:

- **Proxy server** – Specifies which web server to use to proxy connections to Node.js. By default, NGINX is used. If you select **none**, static file mappings don't take effect, and GZIP compression is disabled.
- **Node.js version** – Specifies the version of Node.js. For a list of supported Node.js versions, see Node.js in the *AWS Elastic Beanstalk Platforms* guide.
- **GZIP compression** – Specifies whether GZIP compression is enabled. By default, GZIP compression is enabled.
- **Node command** – Lets you enter the command used to start the Node.js application. An empty string (the default) means Elastic Beanstalk uses `app.js`, then `server.js`, and then `npm start`.

Node.js configuration namespace

You can use a configuration file to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be platform specific or apply to all platforms in the Elastic Beanstalk service as a whole. Configuration options are organized into *namespaces*.

You can choose the proxy to use on the instances for your environment by using the `aws:elasticbeanstalk:environment:proxy` namespace. The following example configures your environment to use the Apache HTTPD proxy server.

Example .ebextensions/nodejs-settings.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: apache
```

You can configure the proxy to serve static files by using the `aws:elasticbeanstalk:environment:proxy:staticfiles` namespace. For more information and an example, see the section called “Static files”.

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See Configuration options for more information.

The Amazon Linux AMI (preceding Amazon Linux 2) Node.js platform

If your Elastic Beanstalk Node.js environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), consider the specific configurations and recommendations in this section.

Notes

- The information in this topic only applies to platform branches based on Amazon Linux AMI (AL1). AL2023/AL2 platform branches are incompatible with previous Amazon Linux AMI (AL1) platform versions and *require different configuration settings*.
- On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information about migrating to a current and fully supported Amazon Linux 2023 platform branch, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

Node.js platform-specific configuration options — Amazon Linux AMI (AL1)

Elastic Beanstalk supports some platform-specific configurations options for Amazon Linux AMI Node.js platform versions. You can choose which proxy server to run in front of your application, choose a specific version of Node.js to run, and choose the command used to run your application.

For proxy server, you can use an NGINX or Apache proxy server. You can set the none value to the `ProxyServer` option. With this setting, Elastic Beanstalk runs your application as standalone, not behind any proxy server. If your environment runs a standalone application, update your code to listen to the port that NGINX forwards traffic to.

```
var port = process.env.PORT || 8080;

app.listen(port, function() {
  console.log('Server running at http://127.0.0.1:%s', port);
});
```

Node.js language versions — Amazon Linux AMI (AL1)

In terms of supported language version, the Node.js Amazon Linux AMI platform is different to other Elastic Beanstalk managed platforms. This is because each Node.js platform version supports only a few Node.js language versions. For a list of supported Node.js versions, see Node.js in the *AWS Elastic Beanstalk Platforms* guide.

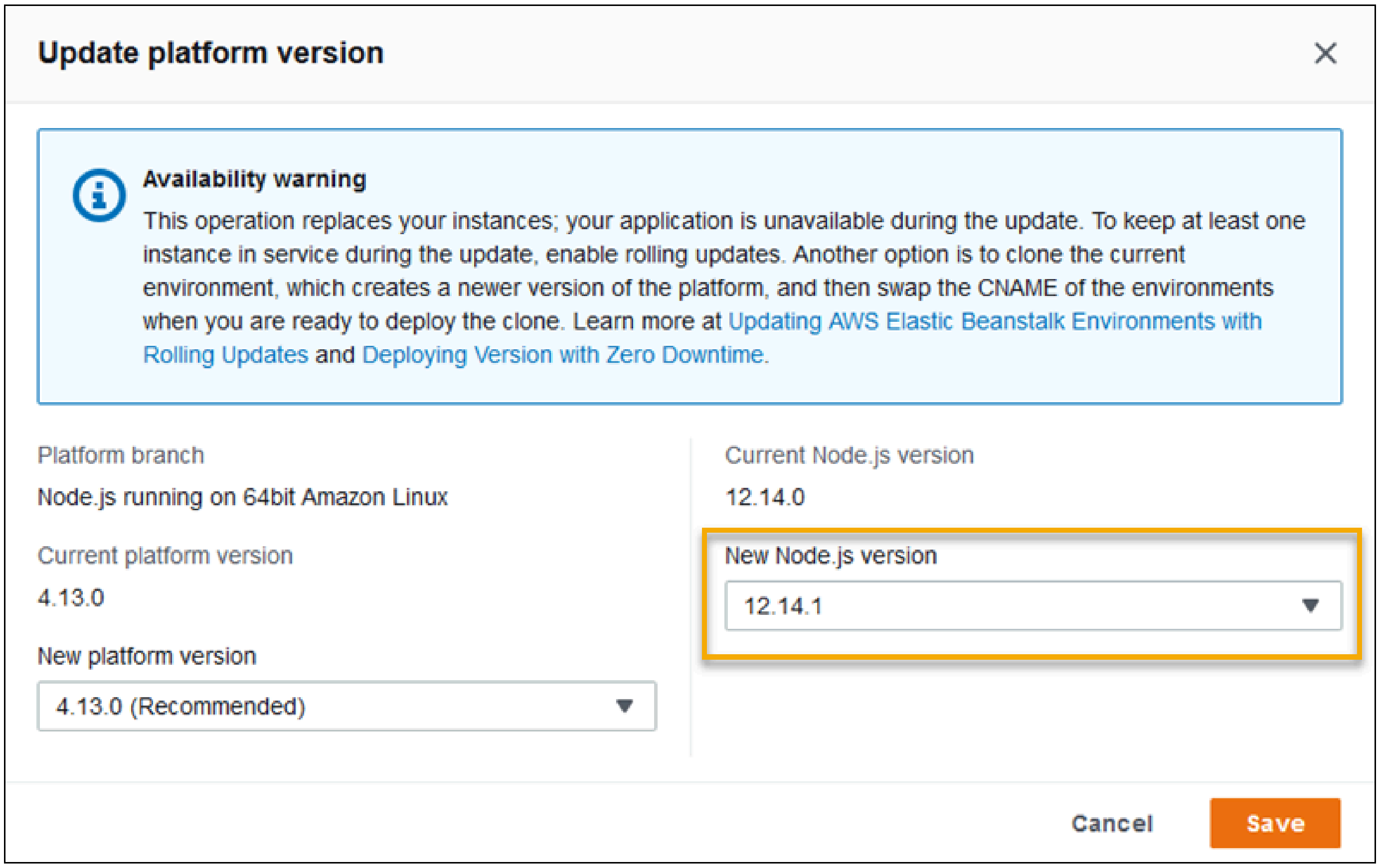
You can use a platform-specific configuration option to set the language version. For instructions, see the section called "Configuring your Node.js environment". Alternatively, use the Elastic Beanstalk console to update the Node.js version that your environment uses as part of updating your platform version.

Note

When support for the version of Node.js that you are using is removed from the platform, you must change or remove the version setting prior to doing a platform update. This might occur when a security vulnerability is identified for one or more versions of Node.js. When this happens, attempting to update to a new version of the platform that doesn't support the configured NodeVersion fails. To avoid needing to create a new environment, change the *NodeVersion* configuration option to a Node.js version that is supported by both the old platform version and the new one, or remove the option setting, and then perform the platform update.

To configure your environment's Node.js version in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the environment overview page, under **Platform**, choose **Change**.
4. On the **Update platform version** dialog box, select a Node.js version.



Update platform version

Availability warning

This operation replaces your instances; your application is unavailable during the update. To keep at least one instance in service during the update, enable rolling updates. Another option is to clone the current environment, which creates a newer version of the platform, and then swap the CNAME of the environments when you are ready to deploy the clone. Learn more at Updating AWS Elastic Beanstalk Environments with Rolling Updates and Deploying Version with Zero Downtime.

Platform branch
Node.js running on 64bit Amazon Linux

Current platform version
4.13.0

New platform version
4.13.0 (Recommended)

Current Node.js version
12.14.0

New Node.js version
12.14.1

5. Choose **Save**.

Node.js configuration namespaces — Amazon Linux AMI (AL1)

The Node.js Amazon Linux AMI platform defines additional options in the `aws:elasticbeanstalk:container:nodejs:staticfiles` and `aws:elasticbeanstalk:container:nodejs` namespaces.

The following configuration file tells Elastic Beanstalk to use `npm start` to run the application. It also sets the proxy type to Apache and enables compression. Last, it configures the proxy to serve static files from two source directories. One source is HTML files at the `html` path under the website's root from the `statichtml` source directory. The other source is image files at the `images` path under the website's root from the `staticimages` source directory.

Example .ebextensions/node-settings.config

```
option_settings:
  aws:elasticbeanstalk:container:nodejs:
    NodeCommand: "npm start"
    ProxyServer: apache
    GzipCompression: true
  aws:elasticbeanstalk:container:nodejs:staticfiles:
    /html: statichtml
    /images: staticimages
```

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See Configuration options for more information.

Configuring custom start commands with a Procfile on Elastic Beanstalk

You can include a file that's called `Procfile` at the root of your source bundle to specify the command that starts your application.

Example Procfile

```
web: node index.js
```

For information about `Procfile` usage see Buildfile and Procfile.

Note

This feature replaces the legacy NodeCommand option in the `aws:elasticbeanstalk:container:nodejs` namespace.

Configuring your application's dependencies on Elastic Beanstalk

Your application might have dependencies on some Node.js modules, such as the ones you specify in `require()` statements. These modules are stored in a `node_modules` directory. When your application runs, Node.js loads the modules from this directory. For more information, see Loading from node_modules folders in the Node.js documentation.

You can specify these module dependencies using a `package.json` file. If Elastic Beanstalk detects this file and a `node_modules` directory isn't present, Elastic Beanstalk runs `npm install` as the *webapp* user. The `npm install` command installs the dependencies in the `node_modules` directory, which Elastic Beanstalk creates beforehand. The `npm install` command accesses the packages listed in the `package.json` file from the public npm registry or other locations. For more information, see the npm Docs website.

If Elastic Beanstalk detects the `node_modules` directory, Elastic Beanstalk doesn't run `npm install`, even if a `package.json` file exists. Elastic Beanstalk assumes that the dependency packages are available in the `node_modules` directory for Node.js to access and load.

The following sections provide more information about establishing your Node.js module dependencies for your application.

Note

If you experience any deployment issues when Elastic Beanstalk is running `npm install`, consider an alternate approach. Include the `node_modules` directory with the dependency modules in your application source bundle. Doing so can circumvent issues with installing dependencies from the public npm registry while you investigate the issue. Because the dependency modules are sourced from a local directory, dong this might also help reduce deployment time. For more information, see Including Node.js dependencies in a node_modules directory

Specifying Node.js dependencies with a package.json file

Include a `package.json` file in the root of your project source to specify dependency packages and to provide a start command. When a `package.json` file is present, and a `node_modules` directory isn't present in the root of your project source, Elastic Beanstalk runs `npm install` as the *webapp* user to install dependencies from the public npm registry. Elastic Beanstalk also uses the `start` command to start your application. For more information about the `package.json` file, see Specifying dependencies in a `package.json` file in the *npm Docs* website.

Use the `scripts` keyword to provide a start command. Currently, the `scripts` keyword is used instead of the legacy `NodeCommand` option in the `aws:elasticbeanstalk:container:nodejs` namespace.

Example package.json – Express

```
{
    "name": "my-app",
    "version": "0.0.1",
    "private": true,
    "dependencies": {
      "ejs": "latest",
      "aws-sdk": "latest",
      "express": "latest",
      "body-parser": "latest"
    },
    "scripts": {
      "start": "node app.js"
    }
  }
```

Production mode and dev dependencies

To specify your dependencies in the `package.json` file use the *dependencies* and *devDependencies* attributes. The *dependencies* attribute designates packages required by your application in production. The *devDependencies* attribute designates packages that are only needed for local development and testing.

Elastic Beanstalk runs `npm install` as the *webapp* user with the following commands. The command options vary depending on the npm version included on platform branch that your application runs on.

- npm v6 — Elastic Beanstalk installs dependencies in production mode by default. It uses the command `npm install --production`.
- npm v7 or greater — Elastic Beanstalk omits the *devDependencies*. It uses the command `npm install --omit=dev`.

Both of the commands listed above do not install the packages that are *devDependencies*.

If you need to install the *devDependencies* packages, set the NPM_USE_PRODUCTION environment property to `false`. With this setting we will not use the above options when running npm install. This will result in the *devDependencies* packages being installed.

SSH and HTTPS

Starting with the March 7, 2023 Amazon Linux 2 platform release, you can also use the SSH and HTTPS protocols to retrieve packages from a Git repository. Platform branch Node.js 16 supports both the SSH and HTTPS protocols. Node.js 14 only supports the HTTPS protocol.

Example package.json – Node.js 16 supports both HTTPS and SSH

```
    ...
    "dependencies": {
      "aws-sdk": "https://github.com/aws/aws-sdk-js.git",
      "aws-chime": "git+ssh://git@github.com:aws/amazon-chime-sdk-js.git"
    }
```

Versions and version ranges

Important

The feature to specify version ranges is not available for Node.js platform branches running on AL2023. We only support one Node.js version within a specific Node.js branch on AL2023. If your `package.json` file specifies a version range, we'll ignore it and default to the platform branch version of Node.js.

Use the `engines` keyword in the `package.json` file to specify the Node.js version that you want your application to use. You can also specify a version range using npm notation. For more information about the syntax for version ranges, see Semantic Versioning using npm on the Node.js website. The `engines` keyword in the Node.js `package.json` file replaces the legacy `NodeVersion` option in the `aws:elasticbeanstalk:container:nodejs` namespace.

Example `package.json` – Single Node.js version

```
{
    ...
    "engines": { "node" : "14.16.0" }
  }
```

Example `package.json` – Node.js version range

```
{
    ...
    "engines": { "node" : ">=10 <11" }
  }
```

When a version range is indicated, Elastic Beanstalk installs the latest Node.js version that the platform has available within the range. In this example, the range indicates that the version must be greater than or equal to version 10, but less than version 11. As a result, Elastic Beanstalk installs the latest Node.js version 10.x.y, which is available on the supported platform.

Be aware that you can only specify a Node.js version that corresponds with your platform branch. For example, if you're using the Node.js 16 platform branch, you can only specify a 16.x.y Node.js version. You can use the version range options supported by npm to allow for more flexibility. For valid Node.js versions for each platform branch, see Node.js in the *AWS Elastic Beanstalk Platforms* guide.

Note

When support for the version of Node.js that you are using is removed from the platform, you must change or remove the Node.js version setting prior to doing a platform update. This might occur when a security vulnerability is identified for one or more versions of Node.js.
When this happens, attempting to update to a new version of the platform that doesn't support the configured Node.js version fails. To avoid needing to create a new environment, change the Node.js version setting in `package.json` to a Node.js version that is supported by both the old platform version and the new one. You have the option to specify a Node.js version range that includes a supported version, as described earlier in this topic. You also have the option to remove the setting, and then deploy the new source bundle.

Including Node.js dependencies in a node_modules directory

To deploy dependency packages to environment instances together with your application code, include them in a directory that's named `node_modules` in the root of your project source. For more information, see Downloading and installing packages locally in the *npm Docs* website.

When you deploy a `node_modules` directory to an AL2023/AL2 Node.js platform version, Elastic Beanstalk assumes that you're providing your own dependency packages, and avoids installing dependencies that are specified in a package.json file. Node.js looks for dependencies in the `node_modules` directory. For more information, see Loading from node_modules Folders in the Node.js documentation.

Note

If you experience any deployment issues when Elastic Beanstalk is running `npm install`, consider using the approach described in this topic as a workaround while you investigate the issue.

Locking dependencies with npm shrinkwrap on Elastic Beanstalk

The Node.js platform runs `npm install` as the *webapp* user each time you deploy. When new versions of your dependencies are available, they're installed when you deploy your application, potentially causing the deployment to take a long time.

You can avoid upgrading dependencies by creating an `npm-shrinkwrap.json` file that locks down your application's dependencies to the current version.

```
$ npm install
$ npm shrinkwrap
wrote npm-shrinkwrap.json
```

Include this file in your source bundle to ensure that dependencies are only installed once.

Configuring the proxy server

Elastic Beanstalk can use NGINX or Apache HTTPD as the reverse proxy to map your application to your Elastic Load Balancing load balancer on port 80. The default is NGINX. Elastic Beanstalk provides a default proxy configuration that you can either extend or completely override with your own configuration.

By default, Elastic Beanstalk configures the proxy to forward requests to your application on port 5000. You can override the default port by setting the `PORT` environment property to the port that your main application listens on.

Note

The port that your application listens on doesn't affect the port that the NGINX server listens to receive requests from the load balancer.

Configuring the proxy server on your platform version

All AL2023/AL2 platforms support a uniform proxy configuration feature. For more information about configuring the proxy server on your platform versions running AL2023/AL2, see Reverse proxy configuration.

Configuring the proxy on Amazon Linux AMI (preceding Amazon Linux 2)

If your Elastic Beanstalk Node.js environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the information in this section.

Notes

- The information in this topic only applies to platform branches based on Amazon Linux AMI (AL1). AL2023/AL2 platform branches are incompatible with previous Amazon Linux AMI (AL1) platform versions and *require different configuration settings*.
- On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information about migrating to a current and fully supported Amazon Linux 2023 platform branch, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

Extending and overriding the default proxy configuration — Amazon Linux AMI (AL1)

The Node.js platform uses a reverse proxy to relay requests from port 80 on the instance to your application that's listening on port 8081. Elastic Beanstalk provides a default proxy configuration that you can either extend or completely override with your own configuration.

To extend the default configuration, add `.conf` files to `/etc/nginx/conf.d` with a configuration file. For a specific example, see Terminating HTTPS on EC2 instances running Node.js.

The Node.js platform sets the PORT environment variable to the port that the proxy server passes traffic to. Read this variable in your code to configure the port for your application.

```
var port = process.env.PORT || 3000;

var server = app.listen(port, function () {
    console.log('Server running at http://127.0.0.1:' + port + '/');
});
```

The default NGINX configuration forwards traffic to an upstream server that's named `nodejs` at `127.0.0.1:8081`. It's possible to remove the default configuration and provide your own in a configuration file.

Example .ebextensions/proxy.config

The following example removes the default configuration and adds a custom configuration that forwards traffic to port 5000, instead of 8081.

```
files:
  /etc/nginx/conf.d/proxy.conf:
    mode: "000644"
    owner: root
    group: root
    content: |
      upstream nodejs {
        server 127.0.0.1:5000;
        keepalive 256;
      }

      server {
        listen 8080;

        if ($time_iso8601 ~ "^(\d{4})-(\d{2})-(\d{2})T(\d{2})") {
            set $year $1;
            set $month $2;
            set $day $3;
            set $hour $4;
        }
        access_log /var/log/nginx/healthd/application.log.$year-$month-$day-$hour
 healthd;
        access_log  /var/log/nginx/access.log  main;

        location / {
            proxy_pass  http://nodejs;
            proxy_set_header   Connection "";
            proxy_http_version 1.1;
            proxy_set_header        Host            $host;
            proxy_set_header        X-Real-IP       $remote_addr;
            proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
        }

        gzip on;
        gzip_comp_level 4;
        gzip_types text/html text/plain text/css application/json application/x-
javascript text/xml application/xml application/xml+rss text/javascript;

        location /static {
            alias /var/app/current/static;
        }

      }

  /opt/elasticbeanstalk/hooks/configdeploy/post/99_kill_default_nginx.sh:
    mode: "000755"
    owner: root
    group: root
    content: |
      #!/bin/bash -xe
      rm -f /etc/nginx/conf.d/00_elastic_beanstalk_proxy.conf
      service nginx stop
      service nginx start

container_commands:
  removeconfig:
    command: "rm -f /tmp/deployment/config/
#etc#nginx#conf.d#00_elastic_beanstalk_proxy.conf /etc/nginx/
conf.d/00_elastic_beanstalk_proxy.conf"
```

The example configuration (`/etc/nginx/conf.d/proxy.conf`) uses the default configuration at `/etc/nginx/conf.d/00_elastic_beanstalk_proxy.conf` as a base to include the default server block with compression and log settings, and a static file mapping.

The `removeconfig` command removes the default configuration for the container so that the proxy server uses the custom configuration. Elastic Beanstalk recreates the default configuration when each configuration is deployed. To account for this, in the following example, a post-configuration-deployment hook (`/opt/elasticbeanstalk/hooks/configdeploy/post/99_kill_default_nginx.sh`) is added. This removes the default configuration and restarts the proxy server.

> **Note**
>
> The default configuration might change in future versions of the Node.js platform. Use the latest version of the configuration as a base for your customizations to ensure compatibility.

If you override the default configuration, you must define any static file mappings and GZIP compression. This is because the platform can't apply the standard settings.

More Elastic Beanstalk example applications and tutorials for Node.js

This section provides additional applications and tutorials. The QuickStart for Node.js topic located previously in this topic walks you through launching the sample Node.js application with the EB CLI.

To get started with Node.js applications on AWS Elastic Beanstalk, all you need is an application source bundle to upload as your first application version and to deploy to an environment.

Launching an environment with a sample Node.js application

Elastic Beanstalk provides single page sample applications for each platform as well as more complex examples that show the use of additional AWS resources such as Amazon RDS and language or platform-specific features and APIs.

> **Note**
>
> Follow the steps in the source bundle `README.md` file to deploy it.

Samples

Environment type	Source bundle	Description
Web Server	nodejs.zip	Single page application.

Environment type	Source bundle	Description
		To launch the sample application with the EB CLI, see QuickStart for Node.js. You can also use the Elastic Beanstalk console to launch the sample application. For detailed steps, see Create an example application in the *Getting started* chapter of this guide.
Web Server with Amazon RDS	nodejs-ex ample-express-rds.zip	Hiking log application that uses the Express framework and an Amazon Relational Database Service (RDS). Tutorial
Web Server with Amazon ElastiCache	nodejs-ex ample-express-elas ticache.z ip	Express web application that uses Amazon ElastiCache for clustering. Clustering enhances your web applicati on's high availability, performance, and security. Tutorial
Web Server with DynamoDB, Amazon SNS and Amazon SQS	nodejs-ex ample-dyn amo.zip	Express web site that collects user contact informati on for a new company's marketing campaign. Uses the AWS SDK for JavaScript in Node.js to write entries to a DynamoDB table, and Elastic Beanstalk configuration files to create resources in DynamoDB, Amazon SNS and Amazon SQS. Tutorial

Next steps

After you have an environment running an application, you can deploy a new version of the application or a completely different application at any time. Deploying a new application version is very quick because it doesn't require provisioning or restarting EC2 instances. For details about application deployment, see Deploy a New Version of Your Application.

After you've deployed a sample application or two and are ready to start developing and running Node.js applications locally, see Setting up your Node.js development environment for Elastic Beanstalk to set up a Node.js development environment with all of the tools that you will need.

Deploying a Node.js Express application to Elastic Beanstalk

This section walks you through deploying a sample application to Elastic Beanstalk using the Elastic Beanstalk Command Line Interface (EB CLI) and then updating the application to use the Express framework.

Prerequisites

This tutorial requires the following prerequisites:

- The Node.js runtimes
- The default Node.js package manager software, npm
- The Express command line generator
- The Elastic Beanstalk Command Line Interface (EB CLI)

For details about installing the first three listed components and setting up your local development environment, see Setting up your Node.js development environment for Elastic Beanstalk. For this tutorial, you don't need to install the AWS SDK for Node.js, which is also mentioned in the referenced topic.

For details about installing and configuring the EB CLI, see Install EB CLI with setup script (recommended) and Configure the EB CLI.

Create an Elastic Beanstalk environment

Your application directory

This tutorial uses a directory called `nodejs-example-express-rds` for the application source bundle. Create the `nodejs-example-express-rds` directory for this tutorial.

```
~$ mkdir nodejs-example-express-rds
```

Note

Each tutorial in this chapter uses it's own directory for the application source bundle. The directory name matches the name of the sample application used by the tutorial.

Change your current working directory to `nodejs-example-express-rds`.

```
~$ cd nodejs-example-express-rds
```

Now, let's set up an Elastic Beanstalk environment running the Node.js platform and the sample application. We'll use the Elastic Beanstalk command line interface (EB CLI).

To configure an EB CLI repository for your application and create an Elastic Beanstalk environment running the Node.js platform

1. Create a repository with the eb init command.

   ```
   ~/nodejs-example-express-rds$ eb init --platform node.js --region <region>
   ```

   This command creates a configuration file in a folder named `.elasticbeanstalk` that specifies settings for creating environments for your application, and creates an Elastic Beanstalk application named after the current folder.

2. Create an environment running a sample application with the eb create command.

   ```
   ~/nodejs-example-express-rds$ eb create --sample nodejs-example-express-rds
   ```

   This command creates a load-balanced environment with the default settings for the Node.js platform and the following resources:

   - **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

     Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or NGINX as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name** – A domain name that routes to your web app in the form *`subdomain`*.*`region`*.*elasticbeanstalk.com*.

  Domain security

  To augment the security of your Elastic Beanstalk applications, the *elasticbeanstalk.com* domain is registered in the Public Suffix List (PSL).
  If you ever need to set sensitive cookies in the default domain name for your Elastic Beanstalk applications, we recommend that you use cookies with a `__Host-` prefix for increased security. This practice defends your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

3. When environment creation completes, use the **eb open** command to open the environment's URL in the default browser.

```
~/nodejs-example-express-rds$ eb open
```

You have now created a Node.js Elastic Beanstalk environment with a sample application. You can update it with your own application. Next, we update the sample application to use the Express framework.

Update the application to use Express

After you've created an environment with a sample application, you can update it with your own application. In this procedure, we first run the **express** and **npm install** commands to set up the Express framework in your application directory. Then we use the EB CLI to update your Elastic Beanstalk environment with the updated application.

To update your application to use Express

1. Run the `express` command. This generates `package.json`, `app.js`, and a few directories.

```
~/nodejs-example-express-rds$ express
```

   When prompted, type **y** if you want to continue.

> **Note**
>
> If the **express** command doesn't work, you may not have installed the Express command line generator as described in the earlier *Prerequisites* section. Or the directory path setting for your local machine may need to be set up to run the **express** command. See the *Prerequisites* section for detailed steps about setting up your development environment, so you can proceed with this tutorial.

2. Set up local dependencies.

```
~/nodejs-example-express-rds$ npm install
```

3. (Optional) Verify the web app server starts up.

```
~/nodejs-example-express-rds$ npm start
```

You should see output similar to the following:

```
> nodejs@0.0.0 start /home/local/user/node-express
> node ./bin/www
```

The server runs on port 3000 by default. To test it, run `curl http://localhost:3000` in another terminal, or open a browser on the local computer and enter URL address `http://localhost:3000`.

Press **Ctrl+C** to stop the server.

4. Deploy the changes to your Elastic Beanstalk environment with the **eb deploy** command.

```
~/nodejs-example-express-rds$ eb deploy
```

5. Once the environment is green and ready, refresh the URL to verify it worked. You should see a web page that says **Welcome to Express**.

Next, let's update the Express application to serve static files and add a new page.

To configure static files and add a new page to your Express application

1. Add a second configuration file in the `.ebextensions` folder with the following content:

`nodejs-example-express-rds/.ebextensions/staticfiles.config`

```
option_settings:
    aws:elasticbeanstalk:environment:proxy:staticfiles:
        /stylesheets: public/stylesheets
```

This setting configures the proxy server to serve files in the `public` folder at the `/public` path of the application. Serving files statically from the proxy server reduces the load on your application. For more information, see Static files earlier in this chapter.

2. (Optional) To confirm that static mappings are configured correctly, comment out the static mapping configuration in `nodejs-example-express-rds/app.js`. This removes the mapping from the node application.

```
//  app.use(express.static(path.join(__dirname, 'public')));
```

The static file mappings in the `staticfiles.config` file from the previous step should still load the stylesheet successfully, even after you comment this line out. To verify that the static file mappings are loaded through the proxy static file configuration, rather than the express application, remove the values following `option_settings:`. After it has been removed from both the static file configuration and the node application, the stylesheet will fail to load.

Remember to reset the contents of both the `nodejs-example-express-rds/app.js` and `staticfiles.config` when you're done testing.

3. Add `nodejs-example-express-rds/routes/hike.js`. Type the following:

```
exports.index = function(req, res) {
 res.render('hike', {title: 'My Hiking Log'});
};

exports.add_hike = function(req, res) {
};
```

4. Update `nodejs-example-express-rds/app.js` to include three new lines.

First, add the following line to add a `require` for this route:

```
var hike = require('./routes/hike');
```

Your file should look similar to the following snippet:

```
var express = require('express');
var path = require('path');
var hike = require('./routes/hike');
```

Then, add the following two lines to `nodejs-example-express-rds/app.js` after `var app = express();`

```
app.get('/hikes', hike.index);
app.post('/add_hike', hike.add_hike);
```

Your file should look similar to the following snippet:

```
var app = express();
```

```
app.get('/hikes', hike.index);
app.post('/add_hike', hike.add_hike);
```

5. Copy `nodejs-example-express-rds/views/index.jade` to `nodejs-example-express-rds/views/hike.jade`.

   ```
   ~/nodejs-example-express-rds$ cp views/index.jade views/hike.jade
   ```

6. Deploy the changes with the **eb deploy** command.

   ```
   ~/nodejs-example-express-rds$ eb deploy
   ```

7. Your environment will be updated after a few minutes. After your environment is green and ready, verify it worked by refreshing your browser and appending **`hikes`** at the end of the URL (e.g., `http://node-express-env-syypntcz2q.elasticbeanstalk.com/hikes`).

   You should see a web page titled **My Hiking Log**.

You have now created a web application that uses the Express framework. In the next section, we'll modify the application to use an Amazon Relational Database Service (RDS) to store a hiking log.

Update the application to use Amazon RDS

In this next step we update the application to use Amazon RDS for MySQL.

To update your application to use RDS for MySQL

1. To create an RDS for MySQL database coupled to your Elastic Beanstalk environment, follow the instructions in the Adding a database topic included later in this chapter. Adding a database instance takes about 10 minutes.
2. Update the dependencies section in the `package.json` with the following contents:

   ```
   "dependencies": {
       "async": "^3.2.4",
       "express": "4.18.2",
       "jade": "1.11.0",
       "mysql": "2.18.1",
       "node-uuid": "^1.4.8",
       "body-parser": "^1.20.1",
       "method-override": "^3.0.0",
       "morgan": "^1.10.0",
   ```

```
    "errorhandler": "^1.5.1"
  }
```

3. Run **npm install**.

```
~/nodejs-example-express-rds$ npm install
```

4. Update `app.js` to connect to the database, create a table, and insert a single default hiking log. Every time this app is deployed it will drop the previous hikes table and recreate it.

```
/**
 * Module dependencies.
 */

 const express = require('express')
 , routes = require('./routes')
 , hike = require('./routes/hike')
 , http = require('http')
 , path = require('path')
 , mysql = require('mysql')
 , async = require('async')
 , bodyParser = require('body-parser')
 , methodOverride = require('method-override')
 , morgan = require('morgan')
 , errorhandler = require('errorhandler');

const { connect } = require('http2');

const app = express()

app.set('views', __dirname + '/views')
app.set('view engine', 'jade')
app.use(methodOverride())
app.use(bodyParser.json())
app.use(bodyParser.urlencoded({ extended: true }))
app.use(express.static(path.join(__dirname, 'public')))


app.set('connection', mysql.createConnection({
host: process.env.RDS_HOSTNAME,
user: process.env.RDS_USERNAME,
password: process.env.RDS_PASSWORD,
port: process.env.RDS_PORT}));
```

```
function init() {
 app.get('/', routes.index);
 app.get('/hikes', hike.index);
 app.post('/add_hike', hike.add_hike);
}

const client = app.get('connection');
async.series([
 function connect(callback) {
   client.connect(callback);
   console.log('Connected!');
 },
 function clear(callback) {
   client.query('DROP DATABASE IF EXISTS mynode_db', callback);
 },
 function create_db(callback) {
   client.query('CREATE DATABASE mynode_db', callback);
 },
 function use_db(callback) {
   client.query('USE mynode_db', callback);
 },
 function create_table(callback) {
    client.query('CREATE TABLE HIKES (' +
                        'ID VARCHAR(40), ' +
                        'HIKE_DATE DATE, ' +
                        'NAME VARCHAR(40), ' +
                        'DISTANCE VARCHAR(40), ' +
                        'LOCATION VARCHAR(40), ' +
                        'WEATHER VARCHAR(40), ' +
                        'PRIMARY KEY(ID))', callback);
 },
 function insert_default(callback) {
   const hike = {HIKE_DATE: new Date(), NAME: 'Hazard Stevens',
         LOCATION: 'Mt Rainier', DISTANCE: '4,027m vertical', WEATHER:'Bad', ID:
 '12345'};
   client.query('INSERT INTO HIKES set ?', hike, callback);
 }
], function (err, results) {
 if (err) {
   console.log('Exception initializing database.');
   throw err;
 } else {
   console.log('Database initialization complete.');
   init();
 }
});

module.exports = app
```

5. Add the following content to `routes/hike.js`. This will enable the routes to insert new hiking logs into the *HIKES* database.

```
const uuid = require('node-uuid');
exports.index = function(req, res) {
  res.app.get('connection').query( 'SELECT * FROM HIKES', function(err,
rows) {
    if (err) {
      res.send(err);
    } else {
      console.log(JSON.stringify(rows));
      res.render('hike', {title: 'My Hiking Log', hikes: rows});
  }});
};
exports.add_hike = function(req, res){
  const input = req.body.hike;
  const hike = { HIKE_DATE: new Date(), ID: uuid.v4(), NAME: input.NAME,
  LOCATION: input.LOCATION, DISTANCE: input.DISTANCE, WEATHER: input.WEATHER};
  console.log('Request to log hike:' + JSON.stringify(hike));
  req.app.get('connection').query('INSERT INTO HIKES set ?', hike, function(err) {
      if (err) {
        res.send(err);
      } else {
        res.redirect('/hikes');
      }
   });
};
```

6. Replace the content of `routes/index.js` with the following:

```
/*
 * GET home page.
 */

exports.index = function(req, res){
  res.render('index', { title: 'Express' });
};
```

7. Add the following jade template to `views/hike.jade` to provide the user interface for adding hiking logs.

```
extends layout

block content
  h1= title
  p Welcome to #{title}

  form(action="/add_hike", method="post")
    table(border="1")
      tr
        td Your Name
        td
          input(name="hike[NAME]", type="textbox")
      tr
        td Location
        td
          input(name="hike[LOCATION]", type="textbox")
      tr
        td Distance
        td
          input(name="hike[DISTANCE]", type="textbox")
      tr
        td Weather
        td
          input(name="hike[WEATHER]", type="radio", value="Good")
          | Good
          input(name="hike[WEATHER]", type="radio", value="Bad")
          | Bad
          input(name="hike[WEATHER]", type="radio", value="Seattle", checked)
          | Seattle
      tr
        td(colspan="2")
          input(type="submit", value="Record Hike")

  div
    h3 Hikes
    table(border="1")
      tr
        td Date
        td Name
        td Location
        td Distance
        td Weather
      each hike in hikes
        tr
          td #{hike.HIKE_DATE.toDateString()}
          td #{hike.NAME}
          td #{hike.LOCATION}
          td #{hike.DISTANCE}
          td #{hike.WEATHER}
```

8. Deploy the changes with the **eb deploy** command.

```
~/nodejs-example-express-rds$ eb deploy
```

Clean up

If you're done working with Elastic Beanstalk, you can terminate your environment.

Use the **eb terminate** command to terminate your environment and all of the resources that it contains.

```
~/nodejs-example-express-rds$ eb terminate
The environment "nodejs-example-express-rds-env" and all associated instances will be
 terminated.
To confirm, type the environment name: nodejs-example-express-rds-env
INFO: terminateEnvironment is starting.
...
```

Deploying a Node.js Express application with clustering to Elastic Beanstalk

This tutorial walks you through deploying a sample application to Elastic Beanstalk using the Elastic Beanstalk Command Line Interface (EB CLI), and then updating the application to use the Express framework, Amazon ElastiCache, and clustering. Clustering enhances your web application's high availability, performance, and security. To learn more about Amazon ElastiCache, go to What Is Amazon ElastiCache (Memcached)? in the *Amazon ElastiCache (Memcached) User Guide*.

Note

This example creates AWS resources, which you might be charged for. For more information about AWS pricing, see https://aws.amazon.com/pricing/. Some services are part of the AWS Free Usage Tier. If you are a new customer, you can test drive these services for free. See https://aws.amazon.com/free/ for more information.

Prerequisites

This tutorial requires the following prerequisites:

- The Node.js runtimes
- The default Node.js package manager software, npm
- The Express command line generator
- The Elastic Beanstalk Command Line Interface (EB CLI)

For details about installing the first three listed components and setting up your local development environment, see Setting up your Node.js development environment for Elastic Beanstalk. For this tutorial, you don't need to install the AWS SDK for Node.js, which is also mentioned in the referenced topic.

For details about installing and configuring the EB CLI, see Install EB CLI with setup script (recommended) and Configure the EB CLI.

Create an Elastic Beanstalk environment

Your application directory

This tutorial uses a directory called `nodejs-example-express-elasticache` for the application source bundle. Create the `nodejs-example-express-elasticache` directory for this tutorial.

```
~$ mkdir nodejs-example-express-elasticache
```

Note

Each tutorial in this chapter uses it's own directory for the application source bundle. The directory name matches the name of the sample application used by the tutorial.

Change your current working directory to `nodejs-example-express-elasticache`.

```
~$ cd nodejs-example-express-elasticache
```

Now, let's set up an Elastic Beanstalk environment running the Node.js platform and the sample application. We'll use the Elastic Beanstalk command line interface (EB CLI).

To configure an EB CLI repository for your application and create an Elastic Beanstalk environment running the Node.js platform

1. Create a repository with the **eb init** command.

   ```
   ~/nodejs-example-express-elasticache$ eb init --platform node.js --region <region>
   ```

   This command creates a configuration file in a folder named `.elasticbeanstalk` that specifies settings for creating environments for your application, and creates an Elastic Beanstalk application named after the current folder.

2. Create an environment running a sample application with the **eb create** command.

   ```
   ~/nodejs-example-express-elasticache$ eb create --sample nodejs-example-express-
   elasticache
   ```

   This command creates a load-balanced environment with the default settings for the Node.js platform and the following resources:

   - **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

     Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or NGINX as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group –** An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer –** An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group –** An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group –** An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket –** A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms –** Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack –** Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name –** A domain name that routes to your web app in the form *`subdomain`*.*`region`*.*elasticbeanstalk.com*.

  Domain security

  To augment the security of your Elastic Beanstalk applications, the *elasticbeanstalk.com* domain is registered in the Public Suffix List (PSL).
  If you ever need to set sensitive cookies in the default domain name for your Elastic Beanstalk applications, we recommend that you use cookies with a `__Host-` prefix for increased security. This practice defends your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

3. When environment creation completes, use the **eb open** command to open the environment's URL in the default browser.

```
~/nodejs-example-express-elasticache$ eb open
```

You have now created a Node.js Elastic Beanstalk environment with a sample application. You can update it with your own application. Next, we update the sample application to use the Express framework.

Update the application to use Express

Update the sample application in the Elastic Beanstalk environment to use the Express framework.

You can download the final source code from nodejs-example-express-elasticache.zip.

To update your application to use Express

After you've created an environment with a sample application, you can update it with your own application. In this procedure, we first run the **express** and **npm install** commands to set up the Express framework in your application directory.

1. Run the `express` command. This generates `package.json`, `app.js`, and a few directories.

   ```
   ~/nodejs-example-express-elasticache$ express
   ```

   When prompted, type **y** if you want to continue.

   Note

   If the **express** command doesn't work, you may not have installed the Express command line generator as described in the earlier *Prerequisites* section. Or the directory path setting for your local machine may need to be set up to run the **express** command. See the *Prerequisites* section for detailed steps about setting up your development environment, so you can proceed with this tutorial.

2. Set up local dependencies.

   ```
   ~/nodejs-example-express-elasticache$ npm install
   ```

3. (Optional) Verify the web app server starts up.

```
~/nodejs-example-express-elasticache$ npm start
```

You should see output similar to the following:

```
> nodejs@0.0.0 start /home/local/user/node-express
> node ./bin/www
```

The server runs on port 3000 by default. To test it, run `curl http://localhost:3000` in another terminal, or open a browser on the local computer and enter URL address `http://localhost:3000`.

Press **Ctrl+C** to stop the server.

4. Rename `nodejs-example-express-elasticache/app.js` to `nodejs-example-express-elasticache/express-app.js`.

```
~/nodejs-example-express-elasticache$ mv app.js express-app.js
```

5. Update the line `var app = express();` in `nodejs-example-express-elasticache/express-app.js` to the following:

```
var app = module.exports = express();
```

6. On your local computer, create a file named `nodejs-example-express-elasticache/app.js` with the following code.

```
/**
 * Module dependencies.
 */

 const express = require('express'),
 session = require('express-session'),
 bodyParser = require('body-parser'),
 methodOverride = require('method-override'),
 cookieParser = require('cookie-parser'),
 fs = require('fs'),
 filename = '/var/nodelist',
 app = express();

let MemcachedStore = require('connect-memcached')(session);
```

```
function setup(cacheNodes) {
 app.use(bodyParser.raw());
 app.use(methodOverride());
 if (cacheNodes.length > 0) {
   app.use(cookieParser());

   console.log('Using memcached store nodes:');
   console.log(cacheNodes);

   app.use(session({
     secret: 'your secret here',
     resave: false,
     saveUninitialized: false,
     store: new MemcachedStore({ 'hosts': cacheNodes })
   }));
 } else {
   console.log('Not using memcached store.');
   app.use(session({
     resave: false,
     saveUninitialized: false, secret: 'your secret here'
   }));
 }

 app.get('/', function (req, resp) {
   if (req.session.views) {
     req.session.views++
     resp.setHeader('Content-Type', 'text/html')
     resp.send(`You are session: ${req.session.id}. Views: ${req.session.views}`)
   } else {
     req.session.views = 1
     resp.send(`You are session: ${req.session.id}. No views yet, refresh the page!
`)
   }
 });

 if (!module.parent) {
   console.log('Running express without cluster. Listening on port %d',
 process.env.PORT || 5000)
   app.listen(process.env.PORT || 5000)
 }
}

console.log("Reading elastic cache configuration")
```

```
// Load elasticache configuration.
fs.readFile(filename, 'UTF8', function (err, data) {
 if (err) throw err;

 let cacheNodes = []
 if (data) {
   let lines = data.split('\n');
   for (let i = 0; i < lines.length; i++) {
     if (lines[i].length > 0) {
       cacheNodes.push(lines[i])
     }
   }
 }

 setup(cacheNodes)
});

module.exports = app;
```

7. Replace the contents of the `nodejs-example-express-elasticache/bin/www` file with the following:

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

const app = require('../app');
const cluster = require('cluster');
const debug = require('debug')('nodejs-example-express-elasticache:server');
const http = require('http');
const workers = {},
  count = require('os').cpus().length;

function spawn() {
  const worker = cluster.fork();
  workers[worker.pid] = worker;
  return worker;
}



/**
```

```
 * Get port from environment and store in Express.
 */

const port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

if (cluster.isMaster) {
  for (let i = 0; i < count; i++) {
    spawn();
  }

  // If a worker dies, log it to the console and start another worker.
  cluster.on('exit', function (worker, code, signal) {
    console.log('Worker ' + worker.process.pid + ' died.');
    cluster.fork();
  });

  // Log when a worker starts listening
  cluster.on('listening', function (worker, address) {
    console.log('Worker started with PID ' + worker.process.pid + '.');
  });

} else {
  /**
   * Create HTTP server.
   */

  let server = http.createServer(app);

  /**
   * Event listener for HTTP server "error" event.
   */

  function onError(error) {
    if (error.syscall !== 'listen') {
      throw error;
    }

    const bind = typeof port === 'string'
      ? 'Pipe ' + port
      : 'Port ' + port;

    // handle specific listen errors with friendly messages
    switch (error.code) {
      case 'EACCES':
        console.error(bind + ' requires elevated privileges');
        process.exit(1);
        break;
      case 'EADDRINUSE':
        console.error(bind + ' is already in use');
        process.exit(1);
        break;
      default:
        throw error;
    }
  }

  /**
   * Event listener for HTTP server "listening" event.
   */

  function onListening() {
    const addr = server.address();
    const bind = typeof addr === 'string'
      ? 'pipe ' + addr
      : 'port ' + addr.port;
    debug('Listening on ' + bind);
  }

  /**
   * Listen on provided port, on all network interfaces.
   */

  server.listen(port);
  server.on('error', onError);
  server.on('listening', onListening);
}

/**
 * Normalize a port into a number, string, or false.
 */

function normalizePort(val) {
  const port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
}
```

8. Deploy the changes to your Elastic Beanstalk environment with the **eb deploy** command.

```
~/nodejs-example-express-elasticache$ eb deploy
```

9. Your environment will be updated after a few minutes. Once the environment is green and ready, refresh the URL to verify it worked. You should see a web page that says "Welcome to Express".

You can access the logs for your EC2 instances running your application. For instructions on accessing your logs, see Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment.

Next, let's update the Express application to use Amazon ElastiCache.

To update your Express application to use Amazon ElastiCache

1. On your local computer, create an `.ebextensions` directory in the top-level directory of your source bundle. In this example, we use `nodejs-example-express-elasticache/.ebextensions`.
2. Create a configuration file `nodejs-example-express-elasticache/.ebextensions/elasticache-iam-with-script.config` with the following snippet. For more information about the configuration file, see Node.js configuration namespace. This creates an IAM user with the permissions required to discover the elasticache nodes and writes to a file anytime the cache changes. You can also copy the file from nodejs-example-express-elasticache.zip. For more information on the ElastiCache properties, see Example: ElastiCache.

Note

YAML relies on consistent indentation. Match the indentation level when replacing content in an example configuration file and ensure that your text editor uses spaces, not tab characters, to indent.

```
Resources:
  MyCacheSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: "Lock cache down to webserver access only"
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort:
            Fn::GetOptionSetting:
              OptionName: CachePort
              DefaultValue: 11211
          ToPort:
            Fn::GetOptionSetting:
              OptionName: CachePort
              DefaultValue: 11211
          SourceSecurityGroupName:
            Ref: AWSEBSecurityGroup
  MyElastiCache:
    Type: 'AWS::ElastiCache::CacheCluster'
    Properties:
      CacheNodeType:
        Fn::GetOptionSetting:
          OptionName: CacheNodeType
          DefaultValue: cache.t2.micro
      NumCacheNodes:
        Fn::GetOptionSetting:
          OptionName: NumCacheNodes
          DefaultValue: 1
      Engine:
        Fn::GetOptionSetting:
          OptionName: Engine
          DefaultValue: redis
      VpcSecurityGroupIds:
        -
```

```
          Fn::GetAtt:
            - MyCacheSecurityGroup
            - GroupId
  AWSEBAutoScalingGroup :
    Metadata :
      ElastiCacheConfig :
        CacheName :
          Ref : MyElastiCache
        CacheSize :
           Fn::GetOptionSetting:
             OptionName : NumCacheNodes
             DefaultValue: 1
  WebServerUser :
    Type : AWS::IAM::User
    Properties :
      Path : "/"
      Policies:
        -
          PolicyName: root
          PolicyDocument :
            Statement :
              -
                Effect : Allow
                Action :
                  - cloudformation:DescribeStackResource
                  - cloudformation:ListStackResources
                  - elasticache:DescribeCacheClusters
                Resource : "*"
  WebServerKeys :
    Type : AWS::IAM::AccessKey
    Properties :
      UserName :
        Ref: WebServerUser

Outputs:
  WebsiteURL:
    Description: sample output only here to show inline string function parsing
    Value: |
      http://`{ "Fn::GetAtt" : [ "AWSEBLoadBalancer", "DNSName" ] }`
  MyElastiCacheName:
    Description: Name of the elasticache
    Value:
      Ref : MyElastiCache
  NumCacheNodes:
    Description: Number of cache nodes in MyElastiCache
    Value:
      Fn::GetOptionSetting:
        OptionName : NumCacheNodes
        DefaultValue: 1

files:
  "/etc/cfn/cfn-credentials" :
    content : |
      AWSAccessKeyId=`{ "Ref" : "WebServerKeys" }`
      AWSSecretKey=`{ "Fn::GetAtt" : ["WebServerKeys", "SecretAccessKey"] }`
    mode : "000400"
    owner : root
    group : root

  "/etc/cfn/get-cache-nodes" :
    content : |
      # Define environment variables for command line tools
      export AWS_ELASTICACHE_HOME="/home/ec2-user/elasticache/$(ls /home/ec2-user/
elasticache/)"
      export AWS_CLOUDFORMATION_HOME=/opt/aws/apitools/cfn
      export PATH=$AWS_CLOUDFORMATION_HOME/bin:$AWS_ELASTICACHE_HOME/bin:$PATH
      export AWS_CREDENTIAL_FILE=/etc/cfn/cfn-credentials
      export JAVA_HOME=/usr/lib/jvm/jre

      # Grab the Cache node names and configure the PHP page
      aws cloudformation list-stack-resources --stack `{ "Ref" :
 "AWS::StackName" }` --region `{ "Ref" : "AWS::Region" }` --output text | grep
 MyElastiCache | awk '{print $4}' | xargs -I {} aws elasticache describe-cache-
clusters --cache-cluster-id {} --region `{ "Ref" : "AWS::Region" }` --show-
cache-node-info --output text | grep '^ENDPOINT' | awk '{print $2 ":" $3}' >
 `{ "Fn::GetOptionSetting" : { "OptionName" : "NodeListPath", "DefaultValue" : "/
var/www/html/nodelist" } }`
    mode : "000500"
    owner : root
    group : root

  "/etc/cfn/hooks.d/cfn-cache-change.conf" :
    "content": |
      [cfn-cache-size-change]
      triggers=post.update
      path=Resources.AWSEBAutoScalingGroup.Metadata.ElastiCacheConfig
      action=/etc/cfn/get-cache-nodes
      runas=root
```

```
sources :
  "/home/ec2-user/elasticache" : "https://elasticache-downloads.s3.amazonaws.com/
AmazonElastiCacheCli-latest.zip"

commands:
  make-elasticache-executable:
    command: chmod -R ugo+x /home/ec2-user/elasticache/*/bin/*

packages :
  "yum" :
    "aws-apitools-cfn"  : []

container_commands:
  initial_cache_nodes:
    command: /etc/cfn/get-cache-nodes
```

3. On your local computer, create a configuration file `nodejs-example-express-elasticache/.ebextensions/elasticache_settings.config` with the following snippet to configure ElastiCache.

```
option_settings:
  "aws:elasticbeanstalk:customoption":
     CacheNodeType: cache.t2.micro
     NumCacheNodes: 1
     Engine: memcached
     NodeListPath: /var/nodelist
```

4. On your local computer, replace `nodejs-example-express-elasticache/express-app.js` with the following snippet. This file reads the nodes list from disk (`/var/nodelist`) and configures express to use `memcached` as a session store if nodes are present. Your file should look like the following.

```
/**
 * Module dependencies.
 */

var express = require('express'),
    session = require('express-session'),
    bodyParser = require('body-parser'),
    methodOverride = require('method-override'),
    cookieParser = require('cookie-parser'),
    fs = require('fs'),
    filename = '/var/nodelist',
    app = module.exports = express();

var MemcachedStore = require('connect-memcached')(session);

function setup(cacheNodes) {
  app.use(bodyParser.raw());
  app.use(methodOverride());
  if (cacheNodes) {
      app.use(cookieParser());

      console.log('Using memcached store nodes:');
      console.log(cacheNodes);

      app.use(session({
          secret: 'your secret here',
          resave: false,
          saveUninitialized: false,
          store: new MemcachedStore({'hosts': cacheNodes})
      }));
  } else {
    console.log('Not using memcached store.');
    app.use(cookieParser('your secret here'));
    app.use(session());
  }

  app.get('/', function(req, resp){
  if (req.session.views) {
      req.session.views++
      resp.setHeader('Content-Type', 'text/html')
      resp.write('Views: ' + req.session.views)
      resp.end()
   } else {
      req.session.views = 1
      resp.end('Refresh the page!')
    }
  });

  if (!module.parent) {
      console.log('Running express without cluster.');
      app.listen(process.env.PORT || 5000);
  }
}
```

```
// Load elasticache configuration.
fs.readFile(filename, 'UTF8', function(err, data) {
    if (err) throw err;

    var cacheNodes = [];
    if (data) {
        var lines = data.split('\n');
        for (var i = 0 ; i < lines.length ; i++) {
            if (lines[i].length > 0) {
                cacheNodes.push(lines[i]);
            }
        }
    }
    setup(cacheNodes);
});
```

5. On your local computer, update `package.json` with the following contents:

```
  "dependencies": {
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "express": "~4.16.1",
    "http-errors": "~1.6.3",
    "jade": "~1.11.0",
    "morgan": "~1.9.1",
    "connect-memcached": "*",
    "express-session": "*",
    "body-parser": "*",
    "method-override": "*"
  }
```

6. Run **npm install**.

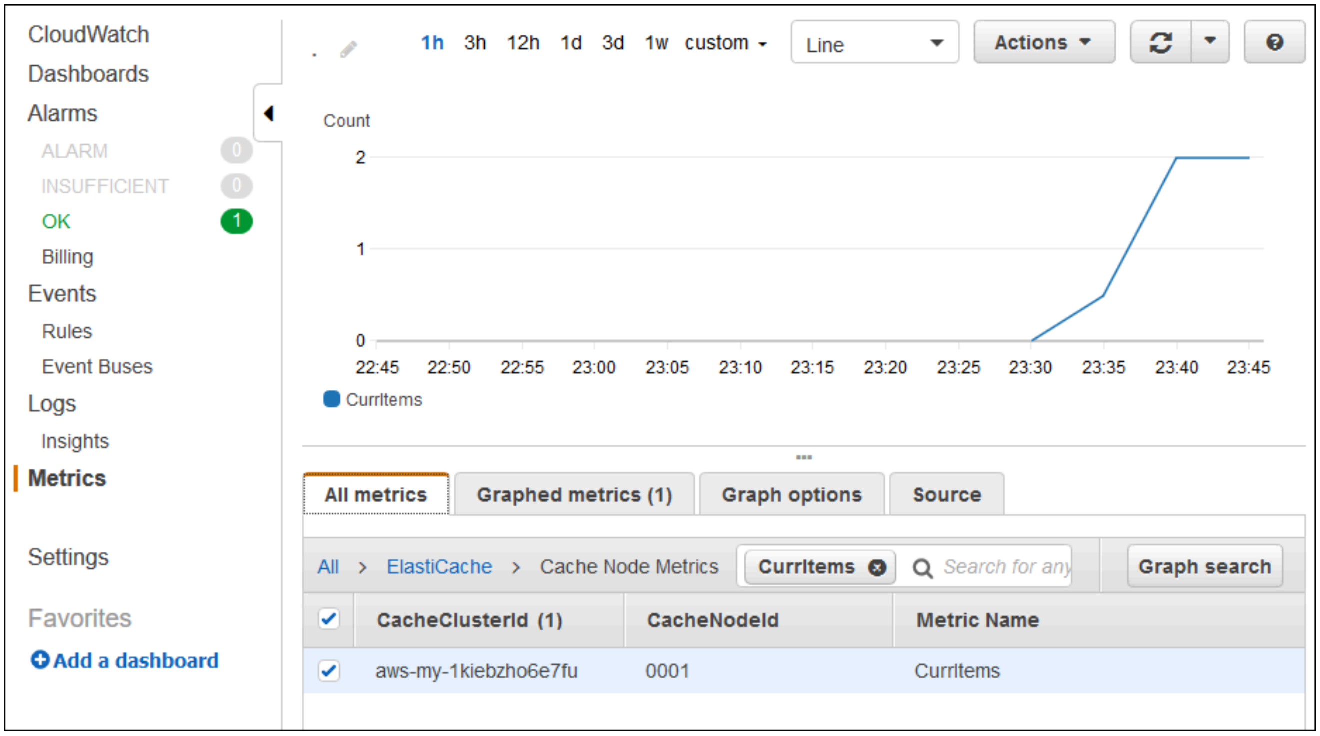
```
~/nodejs-example-express-elasticache$ npm install
```

7. Deploy the updated application.

```
~/nodejs-example-express-elasticache$ eb deploy
```

8. Your environment will be updated after a few minutes. After your environment is green and ready, verify that the code worked.

a. Check the Amazon CloudWatch console to view your ElastiCache metrics. To view your ElastiCache metrics, select **Metrics** in the left pane, and then search for **CurrItems**. Select **ElastiCache > Cache Node Metrics**, and then select your cache node to view the number of items in the cache.



CacheClusterId (1)	CacheNodeId	Metric Name
aws-my-1kiebzho6e7fu	0001	CurrItems

> **Note**
>
> Make sure you are looking at the same region that you deployed your application to.

If you copy and paste your application URL into another web browser and refresh the page, you should see your CurrItem count go up after 5 minutes.

b. Take a snapshot of your logs. For more information about retrieving logs, see Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment.

c. Check the file `/var/log/nodejs/nodejs.log` in the log bundle. You should see something similar to the following:

```
Using memcached store nodes:
[ 'aws-my-1oys9co8zt1uo.1iwtrn.0001.use1.cache.amazonaws.com:11211' ]
```

Clean up

If you no longer want to run your application, you can clean up by terminating your environment and deleting your application.

Use the `eb terminate` command to terminate your environment and the `eb delete` command to delete your application.

To terminate your environment

From the directory where you created your local repository, run `eb terminate`.

```
$ eb terminate
```

This process can take a few minutes. Elastic Beanstalk displays a message once the environment is successfully terminated.

Deploying a Node.js application with DynamoDB to Elastic Beanstalk

This tutorial and its example application nodejs-example-dynamo.zip walks you through the process of deploying a Node.js application that uses the AWS SDK for JavaScript in Node.js to interact with the Amazon DynamoDB service. You'll create a DynamoDB table that's in a database that's decoupled, or external, from the AWS Elastic Beanstalk environment. You'll also configure the application to use a decoupled database. In a production environment, it's best practice to use a database that's decoupled from the Elastic Beanstalk environment so that it's independent from the environment's life cycle. This practice also enables you to perform blue/green deployments.

The example application illustrates the following:

- A DynamoDB table that stores user-provided text data.
- The configuration files to create the table.
- An Amazon Simple Notification Service topic.
- Use of a package.json file to install packages during deployment.

Sections

Prerequisites

This tutorial requires the following prerequisites:

- The Node.js runtimes
- The default Node.js package manager software, npm
- The Express command line generator
- The Elastic Beanstalk Command Line Interface (EB CLI)

For details about installing the first three listed components and setting up your local development environment, see Setting up your Node.js development environment for Elastic Beanstalk. For this tutorial, you don't need to install the AWS SDK for Node.js, which is also mentioned in the referenced topic.

For details about installing and configuring the EB CLI, see Install EB CLI with setup script (recommended) and Configure the EB CLI.

Create an Elastic Beanstalk environment

Your application directory

This tutorial uses a directory called `nodejs-example-dynamo` for the application source bundle. Create the `nodejs-example-dynamo` directory for this tutorial.

```
~$ mkdir nodejs-example-dynamo
```

Note

Each tutorial in this chapter uses it's own directory for the application source bundle. The directory name matches the name of the sample application used by the tutorial.

Change your current working directory to `nodejs-example-dynamo`.

```
~$ cd nodejs-example-dynamo
```

Now, let's set up an Elastic Beanstalk environment running the Node.js platform and the sample application. We'll use the Elastic Beanstalk command line interface (EB CLI).

To configure an EB CLI repository for your application and create an Elastic Beanstalk environment running the Node.js platform

1. Create a repository with the eb init command.

   ```
   ~/nodejs-example-dynamo$ eb init --platform node.js --region <region>
   ```

   This command creates a configuration file in a folder named `.elasticbeanstalk` that specifies settings for creating environments for your application, and creates an Elastic Beanstalk application named after the current folder.

2. Create an environment running a sample application with the eb create command.

   ```
   ~/nodejs-example-dynamo$ eb create --sample nodejs-example-dynamo
   ```

   This command creates a load-balanced environment with the default settings for the Node.js platform and the following resources:

   - **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

     Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or NGINX as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name** – A domain name that routes to your web app in the form `subdomain`.`region`.*elasticbeanstalk.com*.

  Domain security

  To augment the security of your Elastic Beanstalk applications, the *elasticbeanstalk.com* domain is registered in the Public Suffix List (PSL).
  If you ever need to set sensitive cookies in the default domain name for your Elastic Beanstalk applications, we recommend that you use cookies with a `__Host-` prefix for increased security. This practice defends your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

3. When environment creation completes, use the **eb open** command to open the environment's URL in the default browser.

```
~/nodejs-example-dynamo$ eb open
```

You have now created a Node.js Elastic Beanstalk environment with a sample application. You can update it with your own application. Next, we update the sample application to use the Express framework.

Add permissions to your environment's instances

Your application runs on one or more EC2 instances behind a load balancer, serving HTTP requests from the Internet. When it receives a request that requires it to use AWS services, the application uses the permissions of the instance it runs on to access those services.

The sample application uses instance permissions to write data to a DynamoDB table, and to send notifications to an Amazon SNS topic with the SDK for JavaScript in Node.js. Add the following managed policies to the default instance profile to grant the EC2 instances in your environment permission to access DynamoDB and Amazon SNS:

- **AmazonDynamoDBFullAccess**
- **AmazonSNSFullAccess**

To add policies to the default instance profile

1. Open the Roles page in the IAM console.
2. Choose **aws-elasticbeanstalk-ec2-role**.
3. On the **Permissions** tab, choose **Attach policies**.
4. Select the managed policy for the additional services that your application uses. For this tutorial, select `AmazonSNSFullAccess` and `AmazonDynamoDBFullAccess`.
5. Choose **Attach policy**.

See Managing Elastic Beanstalk instance profiles for more on managing instance profiles.

Deploy the example application

Now your environment is ready for you to deploy and run the example application for this tutorial: nodejs-example-dynamo.zip .

To deploy and run the tutorial example application

1. Change your current working directory to the application directory `nodejs-example-dynamo`.

   ```
   ~$ cd nodejs-example-dynamo
   ```

2. Download and extract the contents of the example application source bundle nodejs-example-dynamo.zip to the application directory `nodejs-example-dynamo`.

3. Deploy the example application to your Elastic Beanstalk environment with the **eb deploy** command.

   ```
   ~/nodejs-example-dynamo$ eb deploy
   ```

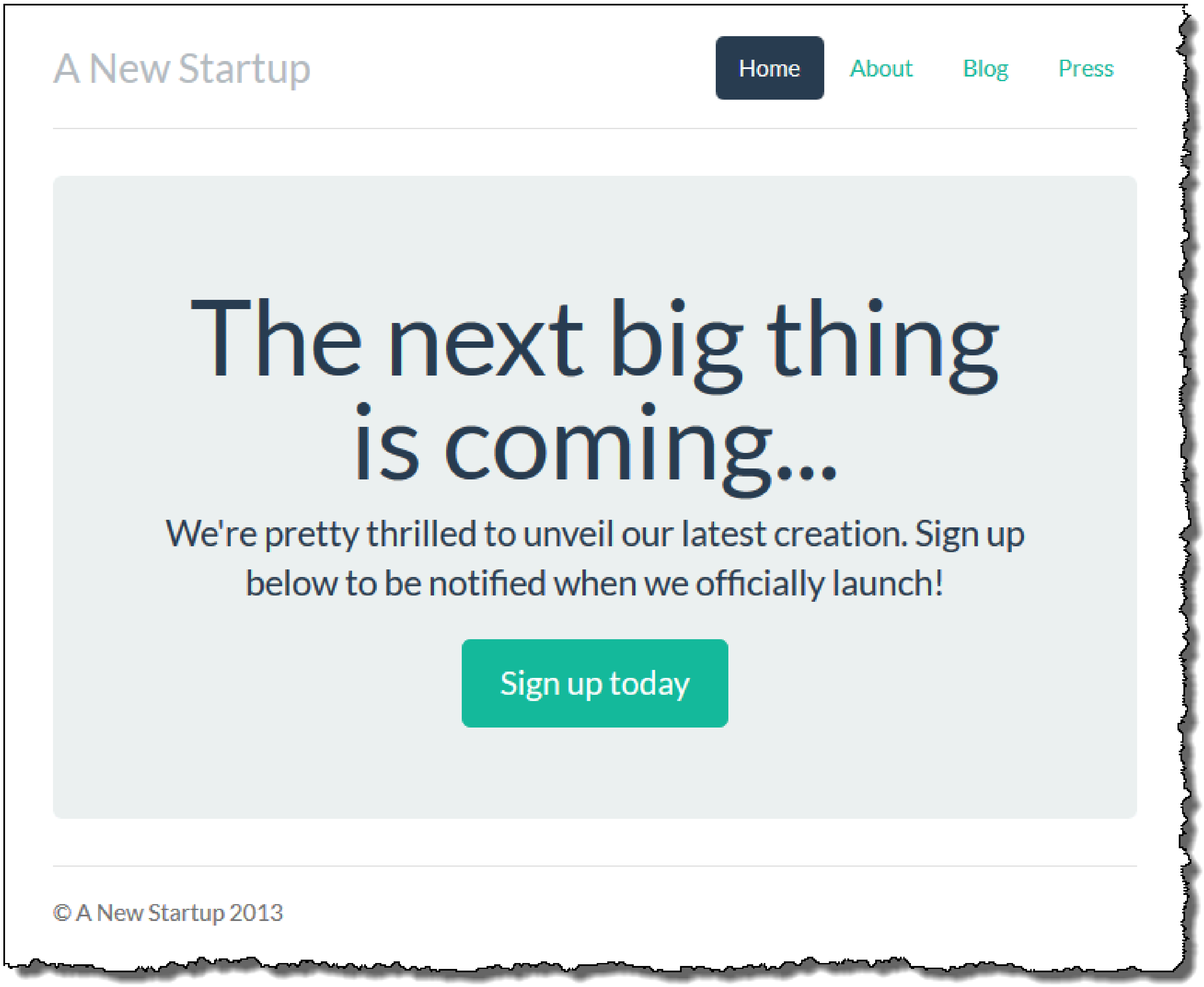
   Note

   By default, the `eb deploy` command creates a ZIP file of your project folder. You can configure the EB CLI to deploy an artifact from your build process instead of creating a ZIP file of your project folder. For more information, see Deploying an artifact instead of the project folder.

4. When environment creation completes, use the **eb open** command to open the environment's URL in the default browser.

   ```
   ~/nodejs-example-dynamo$ eb open
   ```

The site collects user contact information and uses a DynamoDB table to store the data. To add an entry, choose **Sign up today**, enter a name and email address, and then choose **Sign Up!**. The web app writes the form contents to the table and triggers an Amazon SNS email notification.


A New Startup

The next big thing is coming...

We're pretty thrilled to unveil our latest creation. Sign up below to be notified when we officially launch!

© A New Startup 2013

Right now, the Amazon SNS topic is configured with a placeholder email for notifications. You will update the configuration soon, but in the meantime you can verify the DynamoDB table and Amazon SNS topic in the AWS Management Console.

To view the table

1. Open the Tables page in the DynamoDB console.
2. Find the table that the application created. The name starts with **awseb** and contains **StartupSignupsTable**.
3. Select the table, choose **Items**, and then choose **Start search** to view all items in the table.

The table contains an entry for every email address submitted on the signup site. In addition to writing to the table, the application sends a message to an Amazon SNS topic that has two subscriptions, one for email notifications to you, and another for an Amazon Simple Queue Service queue that a worker application can read from to process requests and send emails to interested customers.

To view the topic

1. Open the Topics page in the Amazon SNS console.
2. Find the topic that the application created. The name starts with **awseb** and contains **NewSignupTopic**.
3. Choose the topic to view its subscriptions.

The application (`app.js`) defines two routes. The root path (/) returns a webpage rendered from an Embedded JavaScript (EJS) template with a form that the user fills out to register their name and email address. Submitting the form sends a POST request with the form data to the `/signup` route, which writes an entry to the DynamoDB table and publishes a message to the Amazon SNS topic to notify the owner of the signup.

The sample application includes configuration files that create the DynamoDB table, Amazon SNS topic, and Amazon SQS queue used by the application. This lets you create a new environment and test the functionality immediately, but has the drawback of tying the DynamoDB table to the environment. For a production environment, you should create the DynamoDB table outside of your environment to avoid losing it when you terminate the environment or update its configuration.

Create a DynamoDB table

To use an external DynamoDB table with an application running in Elastic Beanstalk, first create a table in DynamoDB. When you create a table outside of Elastic Beanstalk, it is completely independent of Elastic Beanstalk and your Elastic Beanstalk environments, and will not be terminated by Elastic Beanstalk.

Create a table with the following settings:

- **Table name** – **`nodejs-tutorial`**
- **Primary key** – **`email`**
- Primary key type – **String**

To create a DynamoDB table

1. Open the Tables page in the DynamoDB management console.
2. Choose **Create table**.
3. Type a **Table name** and **Primary key**.
4. Choose the primary key type.
5. Choose **Create**.

Update the application's configuration files

Update the configuration files in the application source to use the **nodejs-tutorial** table instead of creating a new one.

To update the example application for production use

1. Change your current working directory to the application directory `nodejs-example-dynamo`.

   ```
   ~$ cd nodejs-example-dynamo
   ```

2. Open `.ebextensions/options.config` and change the values of the following settings:

   - **NewSignupEmail** – Your email address.
   - **STARTUP_SIGNUP_TABLE – nodejs-tutorial**

   Example .ebextensions/options.config

   ```
   option_settings:
     aws:elasticbeanstalk:customoption:
       NewSignupEmail: you@example.com
     aws:elasticbeanstalk:application:environment:
       THEME: "flatly"
       AWS_REGION: '`{"Ref" : "AWS::Region"}`'
       STARTUP_SIGNUP_TABLE: nodejs-tutorial
       NEW_SIGNUP_TOPIC: '`{"Ref" : "NewSignupTopic"}`'
     aws:elasticbeanstalk:container:nodejs:
       ProxyServer: nginx
     aws:elasticbeanstalk:container:nodejs:staticfiles:
   ```

```
    /static: /static
  aws:autoscaling:asg:
    Cooldown: "120"
  aws:autoscaling:trigger:
    Unit: "Percent"
    Period: "1"
    BreachDuration: "2"
    UpperThreshold: "75"
    LowerThreshold: "30"
    MeasureName: "CPUUtilization"
```

This applies the following configurations for the application:

- The email address that the Amazon SNS topic uses for notifications is set to your address, or the one you enter in the `options.config` file.
- The **nodejs-tutorial** table will be used instead of the one created by `.ebextensions/create-dynamodb-table.config`.

3. Remove `.ebextensions/create-dynamodb-table.config`.

```
~/nodejs-tutorial$ rm .ebextensions/create-dynamodb-table.config
```

The next time you deploy the application, the table created by this configuration file will be deleted.

4. Deploy the updated application to your Elastic Beanstalk environment with the eb deploy command.

```
~/nodejs-example-dynamo$ eb deploy
```

5. When environment creation completes, use the eb open command to open the environment's URL in the default browser.

```
~/nodejs-example-dynamo$ eb open
```

When you deploy, Elastic Beanstalk updates the configuration of the Amazon SNS topic and deletes the DynamoDB table that it created when you deployed the first version of the application.

Now, when you terminate the environment, the **nodejs-tutorial** table will not be deleted. This lets you perform blue/green deployments, modify configuration files, or take down your website without risking data loss.

Open your site in a browser and verify that the form works as you expect. Create a few entries, and then check the DynamoDB console to verify the table.

To view the table

1. Open the Tables page in the DynamoDB console.
2. Find the **nodejs-tutorial** table.
3. Select the table, choose **Items**, and then choose **Start search** to view all items in the table.

You can also see that Elastic Beanstalk deleted the table that it created previously.

Configure your environment for high availability

Finally, configure your environment's Auto Scaling group with a higher minimum instance count. Run at least two instances at all times to prevent the web servers in your environment from being a single point of failure, and to allow you to deploy changes without taking your site out of service.

To configure your environment's Auto Scaling group for high availability

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. In the **Auto Scaling group** section, set **Min instances** to **2**.
6. To save the changes choose **Apply** at the bottom of the page.

Cleanup

After you finish working with the demo code, you can terminate your environment. Elastic Beanstalk deletes all related AWS resources, such as Amazon EC2 instances, database instances, load balancers, security groups, and alarms.

Removing resources does not delete the Elastic Beanstalk application, so you can create new environments for your application at any time.

To terminate your Elastic Beanstalk environment from the console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

You can also delete the external DynamoDB tables that you created.

To delete a DynamoDB table

1. Open the Tables page in the DynamoDB console.
2. Select a table.
3. Choose **Actions**, and then choose **Delete table**.
4. Choose **Delete**.

Next steps

The example application uses configuration files to configure software settings and create AWS resources as part of your environment. See Advanced environment customization with configuration files (`.ebextensions`) for more information about configuration files and their use.

The example application for this tutorial uses the Express web framework for Node.js. For more information about Express, see the official documentation at expressjs.com.

Finally, if you plan on using your application in a production environment, configure a custom domain name for your environment and enable HTTPS for secure connections.

Adding an Amazon RDS DB instance to your Node.js Elastic Beanstalk environment

This topic provides instructions to create an Amazon RDS using the Elastic Beanstalk console. You can use an Amazon Relational Database Service (Amazon RDS) DB instance to store data gathered and modified by your application. The database can be coupled to your environment and managed by Elastic Beanstalk, or it can be created as decoupled and managed externally by another service. In these instructions the database is coupled to your environment and managed by Elastic Beanstalk. For more information about integrating an Amazon RDS with Elastic Beanstalk, see Adding a database to your Elastic Beanstalk environment.

Sections

Adding a DB instance to your environment

To add a DB instance to your environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Choose a DB engine, and enter a user name and password.
6. To save the changes choose **Apply** at the bottom of the page.

Adding a DB instance takes about 10 minutes. When the environment update is complete, the DB instance's hostname and other connection information are available to your application through the following environment properties:

Property name	Description	Property value
`RDS_HOSTNAME`	The hostname of the DB instance.	On the **Connectivity & security** tab on the Amazon RDS console: **Endpoint**.

Property name	Description	Property value
RDS_PORT	The port where the DB instance accepts connectio ns. The default value varies among DB engines.	On the **Connectivity & security** tab on the Amazon RDS console: **Port**.
RDS_DB_NAME	The database name, **ebdb**.	On the **Configuration** tab on the Amazon RDS console: **DB Name**.
RDS_USERNAME	The username that you configured for your database.	On the **Configuration** tab on the Amazon RDS console: **Master username**.
RDS_PASSWORD	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

For more information about configuring a database instance coupled with an Elastic Beanstalk environment, see Adding a database to your Elastic Beanstalk environment.

Downloading a driver

Add the database driver to your project's `package.json` file under `dependencies`.

Example `package.json` – Express with MySQL

```
{
  "name": "my-app",
  "version": "0.0.1",
  "private": true,
  "dependencies": {
    "ejs": "latest",
    "aws-sdk": "latest",
    "express": "latest",
    "body-parser": "latest",
    "mysql": "latest"
  },
  "scripts": {
    "start": "node app.js"
  }
}
```

Common driver packages for Node.js

- **MySQL** – mysql
- **PostgreSQL** – node-postgres
- **SQL Server** – node-mssql
- **Oracle** – node-oracledb

Connecting to a database

Elastic Beanstalk provides connection information for attached DB instances in environment properties. Use `process.env.VARIABLE` to read the properties and configure a database connection.

Example app.js – MySQL database connection

```
var mysql = require('mysql');

var connection = mysql.createConnection({
  host     : process.env.RDS_HOSTNAME,
  user     : process.env.RDS_USERNAME,
  password : process.env.RDS_PASSWORD,
  port     : process.env.RDS_PORT
});

connection.connect(function(err) {
  if (err) {
    console.error('Database connection failed: ' + err.stack);
    return;
  }

  console.log('Connected to database.');
});

connection.end();
```

For more information about constructing a connection string using node-mysql, see npmjs.org/package/mysql.

Node.js tools and resources

There are several places you can go to get additional help when developing your Node.js applications:

Resource	Description
GitHub	Install the AWS SDK for Node.js using GitHub.
AWS SDK for Node.js (Developer Preview)	One-stop shop for sample code, documentation, tools, and additional resources.

Deploying PHP applications with Elastic Beanstalk

You can deploy a PHP application in a few minutes using the Elastic Beanstalk Command Line Interface (EB CLI) or the Elastic Beanstalk console. To learn how, start with the QuickStart tutorial, then review the advanced examples.

Resources for deploying PHP applications to Elastic Beanstalk

- QuickStart for PHP — Step-by-step instructions to deploy a *Hello World* PHP application using the EB CLI.
- Using the Elastic Beanstalk PHP platform — How to use platform features specifically for PHP.
- Advanced examples — How to tutorials for common PHP frameworks and applications, plus how to add an Amazon RDS database to your environment.

For more info on developing with PHP in AWS, see the following resources:

- GitHub — Install the AWS SDK for PHP using GitHub.
- PHP Developer Center — Tools, docs, and sample code to develop PHP applications on AWS.
- AWS SDK for PHP FAQs — Get answers to commonly asked questions.

QuickStart: Deploy a PHP application to Elastic Beanstalk

In the following tutorial, you'll learn how to create and deploy a sample PHP application to an AWS Elastic Beanstalk environment using the EB CLI.

Warning - Not for production use!

Examples are intended for demonstration only. Do not use example applications in production.

Your AWS account

If you're not already an AWS customer, you need to create an AWS account to use Elastic Beanstalk.

Create an AWS account

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.
2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.
2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Prerequisites

- Elastic Beanstalk Command Line Interface - For installation, see Install EB CLI with setup script (recommended).
- PHP - Install PHP on your local machine by following Installation and Configuration instructions on the PHP website.

Step 1: Create a PHP application

For this quick start, you will create a *Hello World* PHP application.

Create a project directory.

```
~$ mkdir eb-php
~$ cd eb-php
```

Next, create an `index.php` file in the project directory and add the following code.

Example index.php

```
<?php
  echo "Hello from a PHP application running in Elastic Beanstalk!";
?>
```

Step 2: Run your application locally

Use the following command to run your application locally.

```
php -S localhost:5000
```

Open a browser to http://localhost:5000.

You should see your hello message in the browser and log messages in your terminal.

Stop the local server by entering `Control+c`, so you can deploy the Elastic Beanstalk.

Step 3: Initialize and deploy your PHP application

Next, you will deploy your application to an *environment* using the Elastic Beanstalk console or the EB CLI. For this tutorial, you'll use the EB CLI with the interactive option to initialize an environment.

To initialize your environment and create an environment

1. Run the following **init** command.

   ```
   eb init -i
   ```

The init command creates an application interactively. The application name will default to the local folder which is `eb-php`.

For all prompts, except SSH access, accept the defaults to create an environment with the latest PHP platform version. For troubleshooting instances, you can set up SSH access by re-running the `eb init -i`command at a later time, or connect using Amazon EC2 Instance Connect or Session Manager.

2. Create an environment and deploy your application

   Run the following command to create an environment named `blue-env`.

   ```
   eb create blue-env
   ```

   When you run the **eb create** command for the first time, Elastic Beanstalk automatically builds a zip file of your application, called a *source bundle*. Next, Elastic Beanstalk creates an environment with one or more Amazon EC2 instances, and then deploys the application into the environment.

   Deploying your application to Elastic Beanstalk might take up to five minutes.

Step 4: Browse your cloud application

When the process to create your environment completes, your application should be running and listening for requests on port 5000. Connect to your application with the following command:

```
eb open
```

The `eb open` command opens a browser tab to a custom subdomain created for your application.

Step 5: Update and redeploy your application

After you have created an application and deployed to an environment, you can deploy a new version of the application or a different application at any time. Deploying a new application version is faster because it doesn't require provisioning or restarting Amazon EC2 instances.

Update your PHP code to include the REQUEST_TIME value from the server environment:

```
<?php
```

```
  echo "Hello from a PHP application running in Elastic Beanstalk!";

  $timestamp = $_SERVER['REQUEST_TIME'];
  echo '<br/>Request time: ' . date('Y/m/d H:i:s', $timestamp);
?>
```

Redeploy your PHP code to Elastic Beanstalk with the following command:

```
eb deploy
```

When you run **eb deploy**, the EB CLI bundles up the contents of your project directory and deploys it to your environment.

After the deploy finishes, refresh the page or reconnect to your application with `eb open`. You should see your updates. If not, troubleshoot by running your local server again to verify your changes.

Congratulations!

You've created, deployed, and updated a PHP application with Elastic Beanstalk!

Clean up

After you finish working with the demo code, you can terminate your environment. Elastic Beanstalk deletes all related AWS resources, such as Amazon EC2 instances, database instances, load balancers, security groups, and alarms.

Removing resources does not delete the Elastic Beanstalk application, so you can create new environments for your application at any time.

To terminate your Elastic Beanstalk environment from the console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

Alternatively, you can terminate your environment with the EB CLI with the following command:

```
eb terminate
```

Next steps

You can explore your application environment using the Elastic Beanstalk console. For more info, see Explore your environment.

For advanced examples using PHP, see Advanced examples for PHP in Elastic Beanstalk.

Using the Elastic Beanstalk PHP platform

AWS Elastic Beanstalk provides and supports various **platform branches** for different versions of PHP. The platforms support PHP web applications that run stand-alone or under Composer. See PHP in the *AWS Elastic Beanstalk Platforms* document for a full list of supported platform branches.

Elastic Beanstalk provides configuration options that you can use to customize the software that runs on the Amazon EC2 instances in your Elastic Beanstalk environment. You can configure environment variables required by your application, enable log rotation to Amazon S3, map folders in your application source that contain static files to paths served by the proxy server, and set common PHP initialization settings.

Configuration options are available in the Elastic Beanstalk console for modifying the configuration of a running environment. To avoid losing your environment's configuration when you terminate it, you can use saved configurations to save your settings and later apply them to another environment.

To save settings in your source code, you can include configuration files. Settings in configuration files are applied every time you create an environment or deploy your application. You can also use configuration files to install packages, run scripts, and perform other instance customization operations during deployments.

If you use Composer, you can include a `composer.json` file in your source bundle to install packages during deployment.

For advanced PHP configuration and PHP settings that are not provided as configuration options, you can use configuration files to provide an `INI` file that can extend and override the default settings applied by Elastic Beanstalk, or install additional extensions.

Settings applied in the Elastic Beanstalk console override the same settings in configuration files, if they exist. This lets you have default settings in configuration files, and override them with environment-specific settings in the console. For more information about precedence, and other methods of changing settings, see Configuration options.

For details about the various ways you can extend an Elastic Beanstalk Linux-based platform, see the section called “Extending Linux platforms”.

PHP platform topics

Installing the AWS SDK for PHP

If you need to manage AWS resources from within your application, install the AWS SDK for PHP. For example, with the SDK for PHP, you can use Amazon DynamoDB (DynamoDB) to store user and session information without creating a relational database.

To install the SDK for PHP with Composer

```
$ composer require aws/aws-sdk-php
```

For more information, see the AWS SDK for PHP homepage. For instructions, see Install the AWS SDK for PHP.

Considerations for PHP 8.1 on Amazon Linux 2

Read this section if you're using the *PHP 8.1 on Amazon Linux 2* platform branch.

Considerations for PHP 8.1 on Amazon Linux 2

Note

The information in this topic only applies to the *PHP 8.1 on Amazon Linux 2* platform branch. It does not apply to the PHP platform branches based on AL2023. It also does not apply to the *PHP 8.0 Amazon Linux 2* platform branch.

Elastic Beanstalk stores the PHP 8.1 related RPM packages for the *PHP 8.1 on Amazon Linux 2* platform branch on the EC2 instances in a local directory, instead of the Amazon Linux repository. You can use **rpm -i** to install packages. Starting with PHP 8.1 Platform Version 3.5.0, Elastic Beanstalk stores the PHP 8.1 related RPM packages in the following local EC2 directory.

`/opt/elasticbeanstalk/RPMS`

The following example installs the *php-debuginfo* package.

```
$rpm -i /opt/elasticbeanstalk/RPMS/php-debuginfo-8.1.8-1.amzn2.x86_64.rpm
```

The version in the package name will vary according to the actual version that's listed in the EC2 local directory `/opt/elasticbeanstalk/RPMS`. Use the same syntax to install other PHP 8.1 RPM packages.

Expand the following section to display a list of RPM packages we provide.

RPM Packages

The following list provides the RMP packages that the Elastic Beanstalk PHP 8.1 platform provides on Amazon Linux 2. These are located in the local directory `/opt/elasticbeanstalk/RPMS`.

The version numbers *8.1.8-1* and *3.7.0-1* in the listed package names are only an example.

- `php-8.1.8-1.amzn2.x86_64.rpm`
- `php-bcmath-8.1.8-1.amzn2.x86_64.rpm`
- `php-cli-8.1.8-1.amzn2.x86_64.rpm`
- `php-common-8.1.8-1.amzn2.x86_64.rpm`
- `php-dba-8.1.8-1.amzn2.x86_64.rpm`

- `php-dbg-8.1.8-1.amzn2.x86_64.rpm`
- `php-debuginfo-8.1.8-1.amzn2.x86_64.rpm`
- `php-devel-8.1.8-1.amzn2.x86_64.rpm`
- `php-embedded-8.1.8-1.amzn2.x86_64.rpm`
- `php-enchant-8.1.8-1.amzn2.x86_64.rpm`
- `php-fpm-8.1.8-1.amzn2.x86_64.rpm`
- `php-gd-8.1.8-1.amzn2.x86_64.rpm`
- `php-gmp-8.1.8-1.amzn2.x86_64.rpm`
- `php-intl-8.1.8-1.amzn2.x86_64.rpm`
- `php-ldap-8.1.8-1.amzn2.x86_64.rpm`
- `php-mbstring-8.1.8-1.amzn2.x86_64.rpm`
- `php-mysqlnd-8.1.8-1.amzn2.x86_64.rpm`
- `php-odbc-8.1.8-1.amzn2.x86_64.rpm`
- `php-opcache-8.1.8-1.amzn2.x86_64.rpm`
- `php-pdo-8.1.8-1.amzn2.x86_64.rpm`
- `php-pear-1.10.13-1.amzn2.noarch.rpm`
- `php-pgsql-8.1.8-1.amzn2.x86_64.rpm`
- `php-process-8.1.8-1.amzn2.x86_64.rpm`
- `php-pspell-8.1.8-1.amzn2.x86_64.rpm`
- `php-snmp-8.1.8-1.amzn2.x86_64.rpm`
- `php-soap-8.1.8-1.amzn2.x86_64.rpm`
- `php-sodium-8.1.8-1.amzn2.x86_64.rpm`
- `php-xml-8.1.8-1.amzn2.x86_64.rpm`
- `php-pecl-imagick-3.7.0-1.amzn2.x86_64.rpm`
- `php-pecl-imagick-debuginfo-3.7.0-1.amzn2.x86_64.rpm`
- `php-pecl-imagick-devel-3.7.0-1.amzn2.noarch.rpm`

You can use the PEAR and PECL packages to install common extensions. For more information about PEAR, see the PEAR PHP Extension and Application Repository website. For more information about PECL, see the PECL extension website.

The following example commands install the Memcached extensions.

```
$pecl install memcache
```

Or you could also use the following:

```
$pear install pecl/memcache
```

The following example commands install the Redis extensions.

```
$pecl install redis
```

Or you could also use the following:

```
$pear install pecl/redis
```

Configuring your PHP environment

You can use the Elastic Beanstalk console to enable log rotation to Amazon S3, configure variables that your application can read from the environment, and change PHP settings.

To configure your PHP environment in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.

PHP settings

- **Proxy server** – The proxy server to use on your environment instances. By default, nginx is used.
- **Document root** – The folder that contains your site's default page. If your welcome page is not at the root of your source bundle, specify the folder that contains it relative to the root path. For example, `/public` if the welcome page is in a folder named `public`.
- **Memory limit** – The maximum amount of memory that a script is allowed to allocate. For example, `512M`.

- **Zlib output compression –** Set to On to compress responses.
- **Allow URL fopen –** Set to `Off` to prevent scripts from downloading files from remote locations.
- **Display errors –** Set to On to show internal error messages for debugging.
- **Max execution time –** The maximum time in seconds that a script is allowed to run before the environment terminates it.

Log options

The Log Options section has two settings:

- **Instance profile–** Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3 –** Specifies whether log files for your application's Amazon EC2 instances are copied to the Amazon S3 bucket associated with your application.

Static files

To improve performance, you can use the **Static files** section to configure the proxy server to serve static files (for example, HTML or images) from a set of directories inside your web application. For each directory, you set the virtual path to directory mapping. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application.

For details about configuring static files using configuration files or the Elastic Beanstalk console, see the section called "Static files".

Environment properties

The **Environment Properties** section lets you specify environment configuration settings on the Amazon EC2 instances that are running your application. These settings are passed in as key-value pairs to the application.

Your application code can access environment properties by using `$_SERVER` or the `get_cfg_var` function.

```
$endpoint = $_SERVER['API_ENDPOINT'];
```

See Environment variables and other software settings for more information.

Namespaces for configuration

You can use a configuration file to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be platform specific or apply to all platforms in the Elastic Beanstalk service as a whole. Configuration options are organized into *namespaces*.

The following namespaces configure both your proxy service and PHP specific options:

- `aws:elasticbeanstalk:environment:proxy:staticfiles` – configure the environment proxy to serve static files. You define mappings of virtual paths to application directories.
- `aws:elasticbeanstalk:environment:proxy` – specify the environment's proxy server.
- `aws:elasticbeanstalk:container:php:phpini` – configure PHP specific options. This namespace includes `composer_options`, which is not available on the Elastic Beanstalk console. This option sets the custom options to use when installing dependencies using Composer through the `composer.phar install` command. For more information about this command, including available options, see install on the *getcomposer.org* website.

The following example configuration file specifies a static files option that maps a directory named `staticimages` to the path `/images`, and shows settings for each of the options available in the `aws:elasticbeanstalk:container:php:phpini` namespace:

Example .ebextensions/php-settings.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: apache
  aws:elasticbeanstalk:environment:proxy:staticfiles:
    /images: staticimages
  aws:elasticbeanstalk:container:php:phpini:
    document_root: /public
    memory_limit: 128M
    zlib.output_compression: "Off"
    allow_url_fopen: "On"
    display_errors: "Off"
    max_execution_time: 60
    composer_options: vendor/package
```

Note

The `aws:elasticbeanstalk:environment:proxy:staticfiles` namespace isn't defined on Amazon Linux AMI PHP platform branches (preceding Amazon Linux 2).

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See Configuration options for more information.

Installing your Elastic Beanstalk PHP application's dependencies

This topic describes how to configure you application to install other PHP packages that it requires. Your application might have dependencies on other PHP packages. You can configure your application to install these dependencies on the environment's Amazon Elastic Compute Cloud (Amazon EC2) instances. Alternatively, you can include your application's dependencies in the source bundle and deploy them with the application. The following section discuss both of these ways.

Use a Composer file to install dependencies on instances

Use a `composer.json` file in the root of your project source to use composer to install packages that your application requires on your environment's Amazon EC2 instances.

Example composer.json

```
{
    "require": {
        "monolog/monolog": "1.0.*"
    }
}
```

When a `composer.json` file is present, Elastic Beanstalk runs `composer.phar install` to install dependencies. You can add options to append to the command by setting the `composer_options` option in the `aws:elasticbeanstalk:container:php:phpini` namespace.

Include dependencies in source bundle

If your application has a large number of dependencies, installing them might take a long time. This can increase deployment and scaling operations, because dependencies are installed on every new instance.

To avoid the negative impact on deployment time, use Composer in your development environment to resolve dependencies and install them into the `vendor` folder.

To include dependencies in your application source bundle

1. Run the following command:

   ```
   % composer install
   ```

2. Include the generated `vendor` folder in the root of your application source bundle.

When Elastic Beanstalk finds a `vendor` folder on the instance, it ignores the `composer.json` file (even if it exists). Your application then uses dependencies from the `vendor` folder.

Updating Composer on Elastic Beanstalk

This topic describes how to configure Elastic Beanstalk to keep Composer up to date. You may have to update Composer if you see an error when you try to install packages with a Composer file, or if you're unable to use the latest platform version. Between platform updates, you can update Composer in your environment instances through the use of configuration files in your `.ebextensions` folder.

You can self-update Composer with the following configuration.

```
commands:
  01updateComposer:
    command: /usr/bin/composer.phar self-update 2.7.0
```

The following option setting sets the `COMPOSER_HOME` environment variable, which configures the location of the Composer cache.

```
option_settings:
  - namespace: aws:elasticbeanstalk:application:environment
```

```
    option_name: COMPOSER_HOME
    value: /home/webapp/composer-home
```

You can combine both of these in the same configuration file in your `.ebextensions` folder.

Example .ebextensions/composer.config

```
commands:
  01updateComposer:
    command: /usr/bin/composer.phar self-update 2.7.0

option_settings:
  - namespace: aws:elasticbeanstalk:application:environment
    option_name: COMPOSER_HOME
    value: /home/webapp/composer-home
```

Note

Due to updates to the Composer installation in the February 22, 2024, AL2023 platform release and the February 28, 2024, AL2 platform release, the Composer self-update may fail if `COMPOSER_HOME` is set when the self-update executes.
The following combined commands will fail to execute: `export COMPOSER_HOME=/home/webapp/composer-home && /usr/bin/composer.phar self-update 2.7.0`
However, the previous example will work. In the previous example, the option setting for `COMPOSER_HOME` will not be passed to the `01updateComposer` execution, and it will not be set when the self-update command executes.

Important

If you omit the version number from the `composer.phar self-update` command, Composer will update to the latest version available every time you deploy your source code, and when new instances are provisioned by Auto Scaling. This could cause scaling operations and deployments to fail if a version of Composer is released that is incompatible with your application.

For more information about the Elastic Beanstalk PHP Platforms, including the version of Composer, see PHP platform versions in the document *AWS Elastic Beanstalk Platforms*.

Extending php.ini in your Elastic Beanstalk configuration

Use a configuration file with a `files` block to add a `.ini` file to `/etc/php.d/` on the instances in your environment. The main configuration file, `php.ini`, pulls in settings from files in this folder in alphabetical order. Many extensions are enabled by default by files in this folder.

Example .ebextensions/mongo.config

```
files:
  "/etc/php.d/99mongo.ini":
    mode: "000755"
    owner: root
    group: root
    content: |
      extension=mongo.so
```

Advanced examples for PHP in Elastic Beanstalk

To get started with PHP applications on AWS Elastic Beanstalk, you need an application source bundle to upload as your first application version to deploy to an environment.

We recommend the QuickStart for PHP to get started with a simple PHP application deployed with the EB CLI.

Advanced PHP examples

- Adding an Amazon RDS DB instance to your PHP Elastic Beanstalk environment
- Deploying a Laravel application to Elastic Beanstalk
- Deploying a CakePHP application to Elastic Beanstalk
- Deploying a Symfony application to Elastic Beanstalk
- Deploying a high-availability PHP application with an external Amazon RDS database to Elastic Beanstalk
- Deploying a high-availability WordPress website with an external Amazon RDS database to Elastic Beanstalk
- Deploying a high-availability Drupal website with an external Amazon RDS database to Elastic Beanstalk

Adding an Amazon RDS DB instance to your PHP Elastic Beanstalk environment

This topic provides instructions to create an Amazon RDS using the Elastic Beanstalk console. You can use an Amazon Relational Database Service (Amazon RDS) DB instance to store data gathered and modified by your application. The database can be coupled to your environment and managed by Elastic Beanstalk, or it can be created as decoupled and managed externally by another service. In these instructions the database is coupled to your environment and managed by Elastic Beanstalk. For more information about integrating an Amazon RDS with Elastic Beanstalk, see Adding a database to your Elastic Beanstalk environment.

Sections

Adding a DB instance to your environment

To add a DB instance to your environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Choose a DB engine, and enter a user name and password.
6. To save the changes choose **Apply** at the bottom of the page.

Adding a DB instance takes about 10 minutes. When the environment update is complete, the DB instance's hostname and other connection information are available to your application through the following environment properties:

Property name	Description	Property value
`RDS_HOSTNAME`	The hostname of the DB instance.	On the **Connectivity & security** tab on the Amazon RDS console: **Endpoint**.
`RDS_PORT`	The port where the DB instance accepts connectio ns. The default value varies among DB engines.	On the **Connectivity & security** tab on the Amazon RDS console: **Port**.
`RDS_DB_NAME`	The database name, **ebdb**.	On the **Configuration** tab on the Amazon RDS console: **DB Name**.
`RDS_USERNAME`	The username that you configured for your database.	On the **Configuration** tab on the Amazon RDS console: **Master username**.
`RDS_PASSWORD`	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

For more information about configuring a database instance coupled with an Elastic Beanstalk environment, see Adding a database to your Elastic Beanstalk environment.

Downloading a driver

To use PHP Data Objects (PDO) to connect to the database, install the driver that matches the database engine that you chose.

- **MySQL** – `PDO_MYSQL`
- **PostgreSQL** – `PDO_PGSQL`
- **Oracle** – `PDO_OCI`
- **SQL Server** – `PDO_SQLSRV`

For more information, see http://php.net/manual/en/pdo.installation.php.

Connecting to a database with a PDO or MySQLi

You can use $_SERVER[`*VARIABLE*`] to read connection information from the environment.

For a PDO, create a Data Source Name (DSN) from the host, port, and name. Pass the DSN to the constructor for the PDO with the database user name and password.

Example Connect to an RDS database with PDO - MySQL

```
<?php
$dbhost = $_SERVER['RDS_HOSTNAME'];
$dbport = $_SERVER['RDS_PORT'];
$dbname = $_SERVER['RDS_DB_NAME'];
$charset = 'utf8' ;

$dsn = "mysql:host={$dbhost};port={$dbport};dbname={$dbname};charset={$charset}";
$username = $_SERVER['RDS_USERNAME'];
$password = $_SERVER['RDS_PASSWORD'];

$pdo = new PDO($dsn, $username, $password);
?>
```

For other drivers, replace `mysql` with the name of your driver – `pgsql`, `oci`, or `sqlsrv`.

For MySQLi, pass the hostname, user name, password, database name, and port to the `mysqli` constructor.

Example Connect to an RDS database with mysqli_connect()

```
$link = new mysqli($_SERVER['RDS_HOSTNAME'], $_SERVER['RDS_USERNAME'],
 $_SERVER['RDS_PASSWORD'], $_SERVER['RDS_DB_NAME'], $_SERVER['RDS_PORT']);
```

Connecting to a database with Symfony

For Symfony version 3.2 and newer, you can use `%env(`*PROPERTY_NAME*`)%` to set database parameters in a configuration file based on the environment properties set by Elastic Beanstalk.

Example app/config/parameters.yml

```
parameters:
    database_driver:   pdo_mysql
    database_host:     '%env(RDS_HOSTNAME)%'
    database_port:     '%env(RDS_PORT)%'
    database_name:     '%env(RDS_DB_NAME)%'
    database_user:     '%env(RDS_USERNAME)%'
    database_password: '%env(RDS_PASSWORD)%'
```

See External Parameters (Symfony 3.4) for more information.

For earlier versions of Symfony, environment variables are only accessible if they start with `SYMFONY__`. This means that the Elastic Beanstalk-defined environment properties are not accessible, and you must define your own environment properties to pass the connection information to Symfony.

To connect to a database with Symfony 2, create an environment property for each parameter. Then, use `%`*`property.name`*`%` to access the Symfony-transformed variable in a configuration file. For example, an environment property named `SYMFONY__DATABASE__USER` is accessible as `database.user`.

```
    database_user:     "%database.user%"
```

See External Parameters (Symfony 2.8) for more information.

Deploying a Laravel application to Elastic Beanstalk

Laravel is an open source, model-view-controller (MVC) framework for PHP. This tutorial walks you through the process of generating a Laravel application, deploying it to an AWS Elastic Beanstalk environment, and configuring it to connect to an Amazon Relational Database Service (Amazon RDS) database instance.

Sections

- Cleanup
- Next steps

Prerequisites

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in Learn how to get started with Elastic Beanstalk to launch your first Elastic Beanstalk environment.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

Laravel 6 requires PHP 7.2 or later. It also requires the PHP extensions listed in the server requirements topic in the official Laravel documentation. Follow the instructions to install PHP and Composer.

For Laravel support and maintenance information, see the support policy topic on the official Laravel documentation.

Launch an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. Choose the **PHP** platform and accept the default settings and sample code.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link: console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.

3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about 5 minutes and creates the following resources:

- **EC2 instance –** An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

  Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or NGINX as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group –** An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer –** An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group –** An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group –** An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket –** A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms –** Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack –** Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name –** A domain name that routes to your web app in the form *subdomain*.*region*.*elasticbeanstalk.com*.

Domain security

To augment the security of your Elastic Beanstalk applications, the *elasticbeanstalk.com* domain is registered in the Public Suffix List (PSL).
If you ever need to set sensitive cookies in the default domain name for your Elastic Beanstalk applications, we recommend that you use cookies with a `__Host-` prefix for increased security. This practice defends your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Note

The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see Using Elastic Beanstalk with Amazon S3.

Install Laravel and generate a website

Composer can install Laravel and create a working project with one command:

```
~$ composer create-project --prefer-dist laravel/laravel eb-laravel
```

Composer installs Laravel and its dependencies, and generates a default project.

If you run into any issues installing Laravel, go to the installation topic in the official documentation: https://laravel.com/docs/6.x.

Deploy your application

Create a source bundle containing the files created by Composer. The following command creates a source bundle named `laravel-default.zip`. It excludes files in the `vendor` folder, which take up a lot of space and are not necessary for deploying your application to Elastic Beanstalk.

```
~/eb-laravel$ zip ../laravel-default.zip -r * .[^.]* -x "vendor/*"
```

Upload the source bundle to Elastic Beanstalk to deploy Laravel to your environment.

To deploy a source bundle

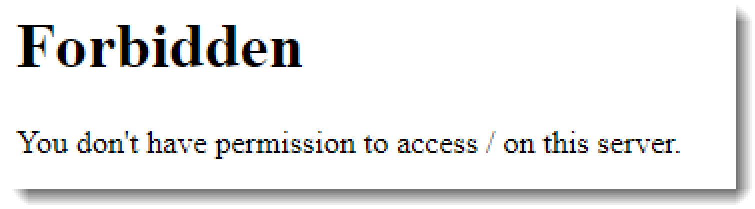
1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Note

To optimize the source bundle further, initialize a Git repository and use the `git archive` command to create the source bundle. The default Laravel project includes a `.gitignore` file that tells Git to exclude the `vendor` folder and other files that are not required for deployment.

Configure Composer settings

When the deployment completes, click the URL to open your Laravel application in the browser:



Forbidden

You don't have permission to access / on this server.

What's this? By default, Elastic Beanstalk serves the root of your project at the root path of the website. In this case, though, the default page (`index.php`) is one level down in the `public` folder. You can verify this by adding `/public` to the URL. For example, `http://`*`laravel.us-east-2`*`.elasticbeanstalk.com/public`.

To serve the Laravel application at the root path, use the Elastic Beanstalk console to configure the *document root* for the website.

To configure your website's document root

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. For **Document Root**, enter **`/public`**.
6. To save the changes choose **Apply** at the bottom of the page.
7. When the update is complete, click the URL to reopen your site in the browser.

So far, so good. Next you'll add a database to your environment and configure Laravel to connect to it.

Add a database to your environment

Launch an RDS DB instance in your Elastic Beanstalk environment. You can use MySQL, SQLServer, or PostgreSQL databases with Laravel on Elastic Beanstalk. For this example, we'll use MySQL.

To add an RDS DB instance to your Elastic Beanstalk environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. For **Engine**, choose **mysql**.
6. Type a master **username** and **password**. Elastic Beanstalk will provide these values to your application using environment properties.
7. To save the changes choose **Apply** at the bottom of the page.

Creating a database instance takes about 10 minutes. For more information about databases coupled to an Elastic Beanstalk environment, see Adding a database to your Elastic Beanstalk environment.

In the meantime, you can update your source code to read connection information from the environment. Elastic Beanstalk provides connection details using environment variables, such as `RDS_HOSTNAME`, that you can access from your application.

Laravel's database configuration is stored in a file named `database.php` in the `config` folder in your project code. Find the `mysql` entry and modify the `host`, `database`, `username`, and `password` variables to read the corresponding values from Elastic Beanstalk:

Example ~/Eb-laravel/config/database.php

```
...
    'connections' => [

        'sqlite' => [
            'driver' => 'sqlite',
            'database' => env('DB_DATABASE', database_path('database.sqlite')),
            'prefix' => '',
        ],

        'mysql' => [
            'driver' => 'mysql',
            'host' => env('RDS_HOSTNAME', '127.0.0.1'),
            'port' => env('RDS_PORT', '3306'),
            'database' => env('RDS_DB_NAME', 'forge'),
            'username' => env('RDS_USERNAME', 'forge'),
            'password' => env('RDS_PASSWORD', ''),
            'unix_socket' => env('DB_SOCKET', ''),
            'charset' => 'utf8mb4',
            'collation' => 'utf8mb4_unicode_ci',
            'prefix' => '',
            'strict' => true,
            'engine' => null,
        ],
...
```

To verify that the database connection is configured correctly, add code to `index.php` to connect to the database and add some code to the default response:

Example ~/Eb-laravel/public/index.php

```
...
if(DB::connection()->getDatabaseName())
{
   echo "Connected to database ".DB::connection()->getDatabaseName();
}
$response->send();
...
```

When the DB instance has finished launching, bundle and deploy the updated application to your environment.

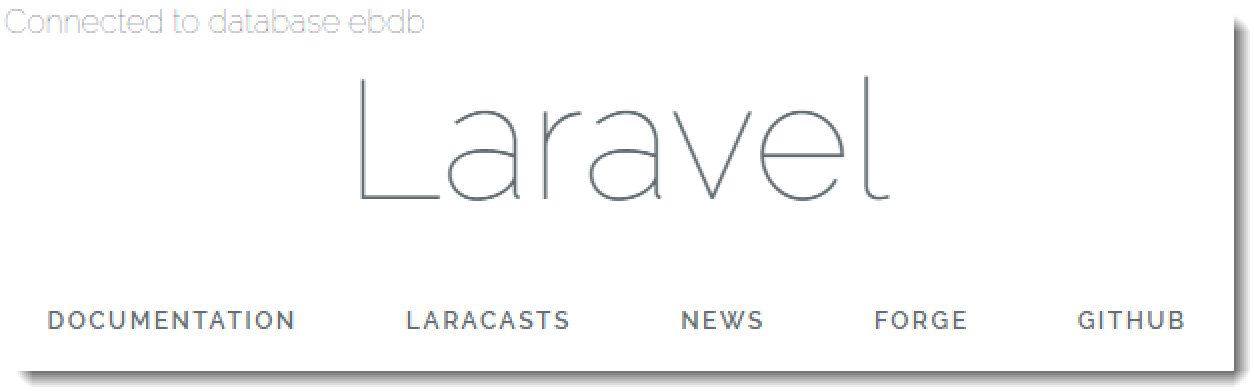
To update your Elastic Beanstalk environment

1. Create a new source bundle:

   ```
   ~/eb-laravel$ zip ../laravel-v2-rds.zip -r * .[^.]* -x "vendor/*"
   ```

2. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
3. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
4. Choose **Upload and Deploy**.
5. Choose **Browse**, and upload `laravel-v2-rds.zip`.
6. Choose **Deploy**.

Deploying a new version of your application takes less than a minute. When the deployment is complete, refresh the web page again to verify that the database connection succeeded:


Connected to database ebdb

Laravel

Cleanup

After you finish working with the demo code, you can terminate your environment. Elastic Beanstalk deletes all related AWS resources, such as Amazon EC2 instances, database instances, load balancers, security groups, and alarms.

Removing resources does not delete the Elastic Beanstalk application, so you can create new environments for your application at any time.

To terminate your Elastic Beanstalk environment from the console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

In addition, you can terminate database resources that you created outside of your Elastic Beanstalk environment. When you terminate an Amazon RDS DB instance, you can take a snapshot and restore the data to another instance later.

To terminate your RDS DB instance

1. Open the Amazon RDS console.
2. Choose **Databases**.
3. Choose your DB instance.
4. Choose **Actions**, and then choose **Delete**.

5. Choose whether to create a snapshot, and then choose **Delete**.

Next steps

For more information about Laravel, go to the Laravel official website at laravel.com.

As you continue to develop your application, you'll probably want a way to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The Elastic Beanstalk Command Line Interface (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

In this tutorial, you used the Elastic Beanstalk console to configure composer options. To make this configuration part of your application source, you can use a configuration file like the following.

Example .ebextensions/composer.config

```
option_settings:
  aws:elasticbeanstalk:container:php:phpini:
    document_root: /public
```

For more information, see Advanced environment customization with configuration files (`.ebextensions`).

Running an Amazon RDS DB instance in your Elastic Beanstalk environment is great for development and testing, but it ties the lifecycle of your database to your environment. See Adding an Amazon RDS DB instance to your PHP Elastic Beanstalk environment for instructions on connecting to a database running outside of your environment.

Finally, if you plan on using your application in a production environment, you will want to configure a custom domain name for your environment and enable HTTPS for secure connections.

Deploying a CakePHP application to Elastic Beanstalk

CakePHP is an open source, MVC framework for PHP. This tutorial walks you through the process of generating a CakePHP project, deploying it to an Elastic Beanstalk environment, and configuring it to connect to an Amazon RDS database instance.

Sections

- Prerequisites

- Launch an Elastic Beanstalk environment
- Install CakePHP and generate a website
- Deploy your application
- Add a database to your environment
- Cleanup
- Next steps

Prerequisites

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in Learn how to get started with Elastic Beanstalk to launch your first Elastic Beanstalk environment.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

CakePHP 4 requires PHP 7.4 or later. It also requires the PHP extensions listed in the official CakePHP installation documentation. You must install both PHP and Composer.

Launch an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. Choose the **PHP** platform and accept the default settings and sample code.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link: console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.

3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about 5 minutes and creates the following resources:

- **EC2 instance –** An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

  Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or NGINX as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group –** An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer –** An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group –** An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group –** An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket –** A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms –** Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack –** Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name –** A domain name that routes to your web app in the form *`subdomain`*.*`region`*.*elasticbeanstalk.com*.

Domain security

To augment the security of your Elastic Beanstalk applications, the *elasticbeanstalk.com* domain is registered in the Public Suffix List (PSL).

If you ever need to set sensitive cookies in the default domain name for your Elastic Beanstalk applications, we recommend that you use cookies with a `__Host-` prefix for increased security. This practice defends your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Note

The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see Using Elastic Beanstalk with Amazon S3.

Install CakePHP and generate a website

Composer can install CakePHP and create a working project with one command:

```
~$ composer create-project --prefer-dist cakephp/app eb-cake
```

Composer installs CakePHP and around 20 dependencies, and generates a default project.

If you run into any issues installing CakePHP, visit the installation topic in the official documentation: http://book.cakephp.org/4.0/en/installation.html

Deploy your application

Create a source bundle containing the files created by Composer. The following command creates a source bundle named `cake-default.zip`. It excludes files in the `vendor` folder, which take up a lot of space and are not necessary for deploying your application to Elastic Beanstalk.

```
eb-cake zip ../cake-default.zip -r * .[^.]* -x "vendor/*"
```

Upload the source bundle to Elastic Beanstalk to deploy CakePHP to your environment.

To deploy a source bundle

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Note

To optimize the source bundle further, initialize a Git repository and use the `git archive` command to create the source bundle. The default Symfony project includes a `.gitignore` file that tells Git to exclude the `vendor` folder and other files that are not required for deployment.

When the process completes, click the URL to open your CakePHP application in the browser.

So far, so good. Next you'll add a database to your environment and configure CakePHP to connect to it.

Add a database to your environment

Launch an Amazon RDS database instance in your Elastic Beanstalk environment. You can use MySQL, SQLServer, or PostgreSQL databases with CakePHP on Elastic Beanstalk. For this example, we'll use PostgreSQL.

To add an Amazon RDS DB instance to your Elastic Beanstalk environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. Under **Database**, choose **Edit**.
5. For **DB engine**, choose **postgres**.
6. Type a master **username** and **password**. Elastic Beanstalk will provide these values to your application using environment properties.
7. To save the changes choose **Apply** at the bottom of the page.

Creating a database instance takes about 10 minutes. In the meantime, you can update your source code to read connection information from the environment. Elastic Beanstalk provides connection details using environment variables such as `RDS_HOSTNAME` that you can access from your application.

CakePHP's database configuration is in a file named `app.php` in the `config` folder in your project code. Open this file and add some code that reads the environment variables from `$_SERVER` and assigns them to local variables. Insert the highlighted lines in the below example after the first line (<?php):

Example ~/Eb-cake/config/app.php

```
<?php
if (!defined('RDS_HOSTNAME')) {
  define('RDS_HOSTNAME', $_SERVER['RDS_HOSTNAME']);
  define('RDS_USERNAME', $_SERVER['RDS_USERNAME']);
  define('RDS_PASSWORD', $_SERVER['RDS_PASSWORD']);
  define('RDS_DB_NAME', $_SERVER['RDS_DB_NAME']);
}
return [
...
```

The database connection is configured further down in `app.php`. Find the following section and modify the default datasources configuration with the name of the driver that matches your database engine (`Mysql`, `Sqlserver`, or `Postgres`), and set the `host`, `username`, `password` and `database` variables to read the corresponding values from Elastic Beanstalk:

Example ~/Eb-cake/config/app.php

```
...
     /**
     * Connection information used by the ORM to connect
     * to your application's datastores.
     * Drivers include Mysql Postgres Sqlite Sqlserver
     * See vendor\cakephp\cakephp\src\Database\Driver for complete list
     */
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Postgres',
            'persistent' => false,
            'host' => RDS_HOSTNAME,
            /*
             * CakePHP will use the default DB port based on the driver selected
             * MySQL on MAMP uses port 8889, MAMP users will want to uncomment
             * the following line and set the port accordingly
             */
            //'port' => 'non_standard_port_number',
            'username' => RDS_USERNAME,
            'password' => RDS_PASSWORD,
            'database' => RDS_DB_NAME,
            /*
             * You do not need to set this flag to use full utf-8 encoding (internal
 default since CakePHP 3.6).
             */
            //'encoding' => 'utf8mb4',
            'timezone' => 'UTC',
            'flags' => [],
            'cacheMetadata' => true,
            'log' => false,
...
```

When the DB instance has finished launching, bundle up and deploy the updated application to your environment:

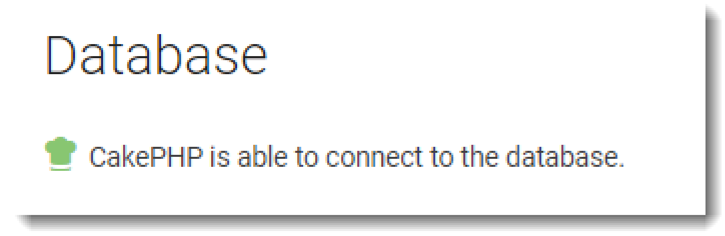
To update your Elastic Beanstalk environment

1. Create a new source bundle:

```
~/eb-cake$ zip ../cake-v2-rds.zip -r * .[^.]* -x "vendor/*"
```

2. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
3. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
4. Choose **Upload and Deploy**.
5. Choose **Browse** and upload `cake-v2-rds.zip`.
6. Choose **Deploy**.

Deploying a new version of your application takes less than a minute. When the deployment is complete, refresh the web page again to verify that the database connection succeeded:



Database

CakePHP is able to connect to the database.

Cleanup

After you finish working with the demo code, you can terminate your environment. Elastic Beanstalk deletes all related AWS resources, such as Amazon EC2 instances, database instances, load balancers, security groups, and alarms.

Removing resources does not delete the Elastic Beanstalk application, so you can create new environments for your application at any time.

To terminate your Elastic Beanstalk environment from the console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

In addition, you can terminate database resources that you created outside of your Elastic Beanstalk environment. When you terminate an Amazon RDS DB instance, you can take a snapshot and restore the data to another instance later.

To terminate your RDS DB instance

1. Open the Amazon RDS console.
2. Choose **Databases**.
3. Choose your DB instance.
4. Choose **Actions**, and then choose **Delete**.
5. Choose whether to create a snapshot, and then choose **Delete**.

Next steps

For more information about CakePHP, read the book at book.cakephp.org.

As you continue to develop your application, you'll probably want a way to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The Elastic Beanstalk Command Line Interface (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

Running an Amazon RDS DB instance in your Elastic Beanstalk environment is great for development and testing, but it ties the lifecycle of your database to your environment. See Adding an Amazon RDS DB instance to your PHP Elastic Beanstalk environment for instructions on connecting to a database running outside of your environment.

Finally, if you plan on using your application in a production environment, you will want to configure a custom domain name for your environment and enable HTTPS for secure connections.

Deploying a Symfony application to Elastic Beanstalk

Symfony is an open-source framework for developing dynamic PHP web applications. This tutorial walks you through the process of generating a Symfony application and deploying it to an AWS Elastic Beanstalk environment.

Sections

- Prerequisites

Prerequisites

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in Learn how to get started with Elastic Beanstalk to launch your first Elastic Beanstalk environment.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

Symfony 4.4.9 requires PHP 7.1.3 or later. It also requires the PHP extensions listed in the technical requirements topic in the official Symfony installation documentation. In this tutorial, we use PHP 7.2 and the corresponding Elastic Beanstalk platform version. Before you proceed, you must install both PHP and Composer.

For Symfony support and maintenance information, see the symfony releases topic on the Symfony website. For more information about updates related to PHP version support for Symfony 4.4.9, see the Symfony 4.4.9 release notes topic on the Symfony website.

Launch an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. Choose the **PHP** platform and accept the default settings and sample code.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link: console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about 5 minutes and creates the following resources:

- **EC2 instance –** An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

  Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or NGINX as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group –** An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer –** An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group –** An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group –** An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket –** A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.

- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name** – A domain name that routes to your web app in the form *`subdomain`.`region`.elasticbeanstalk.com*.

> **Domain security**
>
> To augment the security of your Elastic Beanstalk applications, the *elasticbeanstalk.com* domain is registered in the Public Suffix List (PSL).
> If you ever need to set sensitive cookies in the default domain name for your Elastic Beanstalk applications, we recommend that you use cookies with a `__Host-` prefix for increased security. This practice defends your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

> **Note**
>
> The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see Using Elastic Beanstalk with Amazon S3.

Install Symfony and generate a website

Composer can install Symfony and create a working project with one command:

```
~$ composer create-project symfony/website-skeleton eb-symfony
```

Composer installs Symfony and its dependencies, and generates a default project.

If you run into any issues installing Symfony, go to the installation topic in the official Symfony documentation.

Deploy your application

Go to the project directory.

```
~$ cd eb-symfony
```

Create a source bundle containing the files created by Composer. The following command creates a source bundle named `symfony-default.zip`. It excludes files in the `vendor` folder, which take up a lot of space and are not necessary for deploying your application to Elastic Beanstalk.

```
eb-symfony$ zip ../symfony-default.zip -r * .[^.]* -x "vendor/*"
```

Upload the source bundle to Elastic Beanstalk to deploy Symfony to your environment.

To deploy a source bundle

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Note

To optimize the source bundle further, initialize a Git repository and use the `git archive` command to create the source bundle. The default Symfony project includes a `.gitignore` file that tells Git to exclude the `vendor` folder and other files that are not required for deployment.

Configure Composer settings

When the deployment completes, click the URL to open your Symfony application in the browser.

What's this? By default, Elastic Beanstalk serves the root of your project at the root path of the web site. In this case, though, the default page (`app.php`) is one level down in the `web` folder. You can verify this by adding `/public` to the URL. For example, `http://*symfony.us-east-2*.elasticbeanstalk.com/public`.

To serve the Symfony application at the root path, use the Elastic Beanstalk console to configure the *document root* for the web site.

To configure your web site's document root

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. For **Document root**, enter **`/public`**.
6. To save the changes choose **Apply** at the bottom of the page.
7. When the update is complete, click the URL to reopen your site in the browser.

Cleanup

After you finish working with the demo code, you can terminate your environment. Elastic Beanstalk deletes all related AWS resources, such as Amazon EC2 instances, database instances, load balancers, security groups, and alarms.

Removing resources does not delete the Elastic Beanstalk application, so you can create new environments for your application at any time.

To terminate your Elastic Beanstalk environment from the console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

3. Choose **Actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

Next steps

For more information about Symfony, see What is Symfony? at symfony.com.

As you continue to develop your application, you'll probably want a way to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The Elastic Beanstalk Command Line Interface (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

In this tutorial, you used the Elastic Beanstalk console to configure composer options. To make this configuration part of your application source, you can use a configuration file like the following.

Example .ebextensions/composer.config

```
option_settings:
  aws:elasticbeanstalk:container:php:phpini:
    document_root: /public
```

For more information, see Advanced environment customization with configuration files (`.ebextensions`).

Symfony uses its own configuration files to configure database connections. For instructions on connecting to a database with Symfony, see Connecting to a database with Symfony.

Finally, if you plan on using your application in a production environment, you will want to configure a custom domain name for your environment and enable HTTPS for secure connections.

Deploying a high-availability PHP application with an external Amazon RDS database to Elastic Beanstalk

This tutorial walks you through the process of launching an RDS DB instance external to AWS Elastic Beanstalk, and configuring a high-availability environment running a PHP application to connect to it. Running a DB instance external to Elastic Beanstalk decouples the database from the lifecycle of your environment. This lets you connect to the same database from multiple environments, swap out one database for another, or perform a blue/green deployment without affecting your database.

The tutorial uses a sample PHP application that uses a MySQL database to store user-provided text data. The sample application uses configuration files to configure PHP settings and to create a table in the database for the application to use. It also shows how to use a Composer file to install packages during deployment.

Sections

Prerequisites

Before you start, download the sample application source bundle from GitHub: eb-demo-php-simple-app-1.3.zip

The procedures in this tutorial for Amazon Relational Database Service (Amazon RDS) tasks assume that you are launching resources in a default Amazon Virtual Private Cloud (Amazon VPC). All new accounts include a default VPC in each region. If you don't have a default VPC, the procedures will vary. See Using Elastic Beanstalk with Amazon RDS for instructions for EC2-Classic and custom VPC platforms.

Launch a DB instance in Amazon RDS

To use an external database with an application running in Elastic Beanstalk, first launch a DB instance with Amazon RDS. When you launch an instance with Amazon RDS, it is completely independent of Elastic Beanstalk and your Elastic Beanstalk environments, and will not be terminated or monitored by Elastic Beanstalk.

Use the Amazon RDS console to launch a Multi-AZ **MySQL** DB instance. Choosing a Multi-AZ deployment ensures that your database will fail over and continue to be available if the source DB instance goes out of service.

To launch an RDS DB instance in a default VPC

1. Open the RDS console.
2. In the navigation pane, choose **Databases**.
3. Choose **Create database**.
4. Choose **Standard Create**.

> **Important**
>
> Do not choose **Easy Create**. If you choose it, you can't configure the necessary settings to launch this RDS DB.

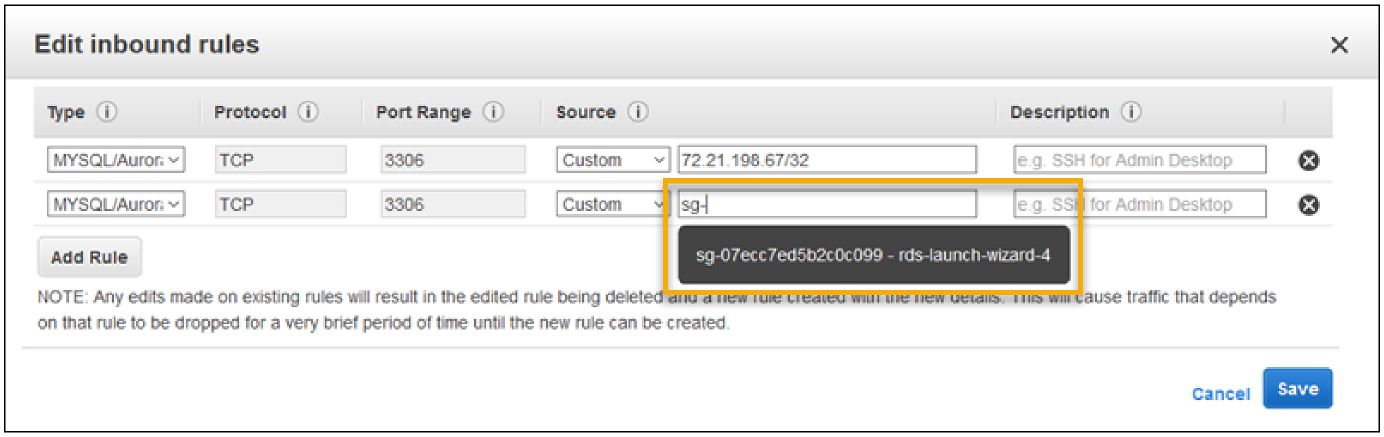
5. Under **Additional configuration**, for **Initial database name**, type **ebdb**.
6. Review the default settings and adjust these settings according to your specific requirements. Pay attention to the following options:
   - **DB instance class** – Choose an instance size that has an appropriate amount of memory and CPU power for your workload.
   - **Multi-AZ deployment** – For high availability, set this to **Create an Aurora Replica/Reader node in a different AZ**.
   - **Master username** and **Master password** – The database username and password. Make a note of these settings because you will use them later.
7. Verify the default settings for the remaining options, and then choose **Create database**.

Next, modify the security group attached to your DB instance to allow inbound traffic on the appropriate port. This is the same security group that you will attach to your Elastic Beanstalk environment later, so the rule that you add will grant ingress permission to other resources in the same security group.

To modify the inbound rules on the security group that's attached to your RDS instance

1. Open the Amazon RDS console.
2. Choose **Databases**.
3. Choose the name of your DB instance to view its details.
4. In the **Connectivity** section, make a note of the **Subnets**, **Security groups**, and **Endpoint** that are displayed on this page. This is so you can use this information later.

5. Under **Security**, you can see the security group that's associated with the DB instance. Open the link to view the security group in the Amazon EC2 console.
6. In the security group details, choose **Inbound**.
7. Choose **Edit**.
8. Choose **Add Rule**.
9. For **Type**, choose the DB engine that your application uses.
10. For **Source**, type **sg-** to view a list of available security groups. Choose the security group that's associated with the Auto Scaling group that's used with your Elastic Beanstalk environment. This is so that Amazon EC2 instances in the environment can have access to the database.



Edit inbound rules

Type	Protocol	Port Range	Source		Description
MYSQL/Auror	TCP	3306	Custom	72.21.198.67/32	e.g. SSH for Admin Desktop
MYSQL/Auror	TCP	3306	Custom	sg-	e.g. SSH for Admin Desktop

sg-07ecc7ed5b2c0c099 - rds-launch-wizard-4

NOTE: Any edits made on existing rules will result in the edited rule being deleted and a new rule created with the new details. This will cause traffic that depends on that rule to be dropped for a very brief period of time until the new rule can be created.

11. Choose **Save**.

Creating a DB instance takes about 10 minutes. In the meantime, create your Elastic Beanstalk environment.

Create an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. Choose the **PHP** platform and accept the default settings and sample code. After you launch the environment, you can configure the environment to connect to the database, then deploy the sample application that you downloaded from GitHub.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link: console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced

2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about 5 minutes and creates the following resources:

- **EC2 instance –** An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

  Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or NGINX as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group –** An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer –** An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group –** An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group –** An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket –** A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms –** Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack –** Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.

- **Domain name** – A domain name that routes to your web app in the form *subdomain.region.elasticbeanstalk.com*.

> **Domain security**
>
> To augment the security of your Elastic Beanstalk applications, the *elasticbeanstalk.com* domain is registered in the Public Suffix List (PSL).
> If you ever need to set sensitive cookies in the default domain name for your Elastic Beanstalk applications, we recommend that you use cookies with a `__Host-` prefix for increased security. This practice defends your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains. The RDS DB instance that you launched is outside of your environment, so you are responsible for managing its lifecycle.

> **Note**
>
> The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see Using Elastic Beanstalk with Amazon S3.

Configure security groups, environment properties, and scaling

Add the security group of your DB instance to your running environment. This procedure causes Elastic Beanstalk to reprovision all instances in your environment with the additional security group attached.

To add a security group to your environment

- Do one of the following:
    - To add a security group using the Elastic Beanstalk console
        a. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.

   b. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
   c. In the navigation pane, choose **Configuration**.
   d. In the **Instances** configuration category, choose **Edit**.
   e. Under **EC2 security groups**, choose the security group to attach to the instances, in addition to the instance security group that Elastic Beanstalk creates.
   f. To save the changes choose **Apply** at the bottom of the page.
   g. Read the warning, and then choose **Confirm**.

- To add a security group using a configuration file, use the `securitygroup-addexisting.config` example file.

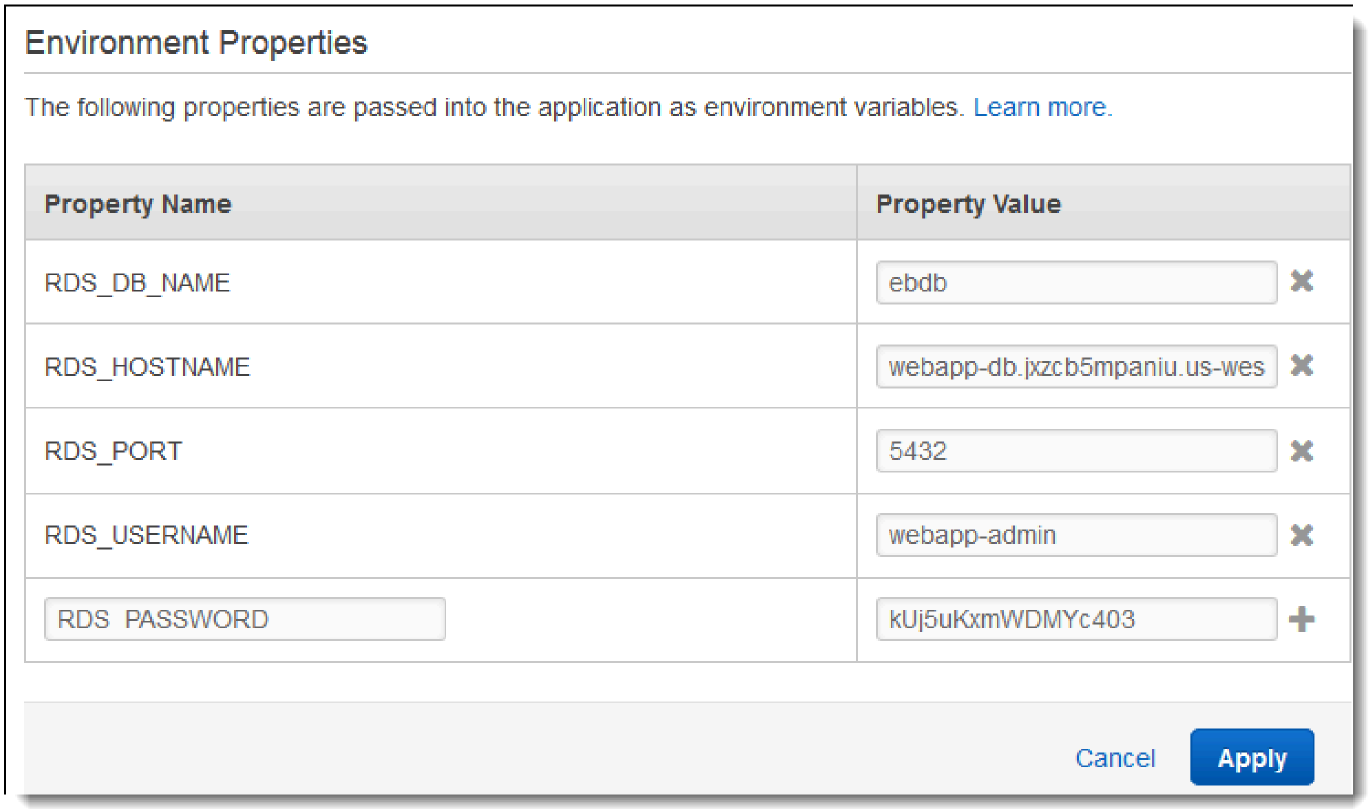
Next, use environment properties to pass the connection information to your environment. The sample application uses a default set of properties that match the ones that Elastic Beanstalk configures when you provision a database within your environment.

To configure environment properties for an Amazon RDS DB instance

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. In the **Environment properties** section, define the variables that your application reads to construct a connection string. For compatibility with environments that have an integrated RDS DB instance, use the following names and values. You can find all values, except for your password, in the RDS console.

Property name	Description	Property value
`RDS_HOSTNAME`	The hostname of the DB instance.	On the **Connectivity & security** tab on the Amazon RDS console: **Endpoint**.

Property name	Description	Property value
`RDS_PORT`	The port where the DB instance accepts connectio ns. The default value varies among DB engines.	On the **Connectivity & security** tab on the Amazon RDS console: **Port**.
`RDS_DB_NAME`	The database name, **ebdb**.	On the **Configuration** tab on the Amazon RDS console: **DB Name**.
`RDS_USERNAME`	The username that you configured for your database.	On the **Configuration** tab on the Amazon RDS console: **Master username**.
`RDS_PASSWORD`	The password that you configured for your database.	Not available for reference in the Amazon RDS console.


Environment Properties

The following properties are passed into the application as environment variables. Learn more.

Property Name	Property Value
RDS_DB_NAME	ebdb
RDS_HOSTNAME	webapp-db.jxzcb5mpaniu.us-wes
RDS_PORT	5432
RDS_USERNAME	webapp-admin
RDS_PASSWORD	kUj5uKxmWDMYc403

6. To save the changes choose **Apply** at the bottom of the page.

Finally, configure your environment's Auto Scaling group with a higher minimum instance count. Run at least two instances at all times to prevent the web servers in your environment from being a single point of failure, and to allow you to deploy changes without taking your site out of service.

To configure your environment's Auto Scaling group for high availability

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. In the **Auto Scaling group** section, set **Min instances** to **2**.
6. To save the changes choose **Apply** at the bottom of the page.

Deploy the sample application

Now your environment is ready to run the sample application and connect to Amazon RDS. Deploy the sample application to your environment.

Note

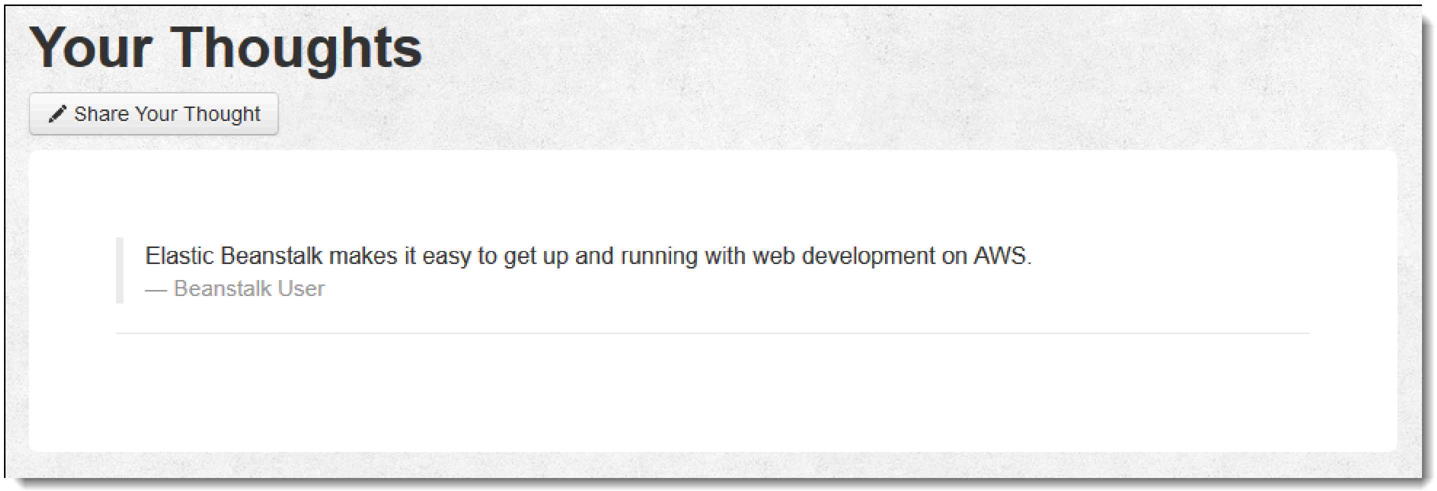
Download the source bundle from GitHub, if you haven't already: eb-demo-php-simple-app-1.3.zip

To deploy a source bundle

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.

6. When the deployment completes, you can choose the site URL to open your website in a new tab.

The site collects user comments and uses a MySQL database to store the data. To add a comment, choose **Share Your Thought**, enter a comment, and then choose **Submit Your Thought**. The web app writes the comment to the database so that any instance in the environment can read it, and it won't be lost if instances go out of service.



Elastic Beanstalk makes it easy to get up and running with web development on AWS.
— Beanstalk User

Cleanup

After you finish working with the demo code, you can terminate your environment. Elastic Beanstalk deletes all related AWS resources, such as Amazon EC2 instances, database instances, load balancers, security groups, and alarms.

Removing resources does not delete the Elastic Beanstalk application, so you can create new environments for your application at any time.

To terminate your Elastic Beanstalk environment from the console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

In addition, you can terminate database resources that you created outside of your Elastic Beanstalk environment. When you terminate an Amazon RDS DB instance, you can take a snapshot and restore the data to another instance later.

To terminate your RDS DB instance

1. Open the Amazon RDS console.
2. Choose **Databases**.
3. Choose your DB instance.
4. Choose **Actions**, and then choose **Delete**.
5. Choose whether to create a snapshot, and then choose **Delete**.

Next steps

As you continue to develop your application, you'll probably want a way to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The Elastic Beanstalk Command Line Interface (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

The sample application uses configuration files to configure PHP settings and create a table in the database if it doesn't already exist. You can also use a configuration file to configure the security group settings of your instances during environment creation to avoid time-consuming configuration updates. See Advanced environment customization with configuration files (`.ebextensions`) for more information.

For development and testing, you might want to use the Elastic Beanstalk functionality for adding a managed DB instance directly to your environment. For instructions on setting up a database inside your environment, see Adding a database to your Elastic Beanstalk environment.

If you need a high-performance database, consider using Amazon Aurora. Amazon Aurora is a MySQL-compatible database engine that offers commercial database features at low cost. To connect your application to a different database, repeat the security group configuration steps and update the RDS-related environment properties.

Finally, if you plan on using your application in a production environment, you will want to configure a custom domain name for your environment and enable HTTPS for secure connections.

Deploying a high-availability WordPress website with an external Amazon RDS database to Elastic Beanstalk

This tutorial describes how to launch an Amazon RDS DB instance that is external to AWS Elastic Beanstalk, then how to configure a high-availability environment running a WordPress website to connect to it. The website uses Amazon Elastic File System (Amazon EFS) as the shared storage for uploaded files.

Running a DB instance external to Elastic Beanstalk decouples the database from the lifecycle of your environment. This lets you connect to the same database from multiple environments, swap out one database for another, or perform a blue/green deployment without affecting your database.

Note

For current information about the compatibility of PHP releases with WordPress versions, see PHP Compatibility and WordPress Versions on the WordPress website. You should refer to this information before you upgrade to a new release of PHP for your WordPress implementations.

Topics

- Next steps

Prerequisites

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in Learn how to get started with Elastic Beanstalk to launch your first Elastic Beanstalk environment.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

Default VPC

The Amazon Relational Database Service (Amazon RDS) procedures in this tutorial assume that you are launching resources in a default Amazon Virtual Private Cloud (Amazon VPC). All new accounts include a default VPC in each AWS Region. If you don't have a default VPC, the procedures will vary. See Using Elastic Beanstalk with Amazon RDS for instructions for EC2-Classic and custom VPC platforms.

AWS Regions

The sample application uses Amazon EFS, which only works in AWS Regions that support Amazon EFS. To learn about supported AWS Regions, see Amazon Elastic File System Endpoints and Quotas in the *AWS General Reference*.

Launch a DB instance in Amazon RDS

When you launch an instance with Amazon RDS, it's completely independent of Elastic Beanstalk and your Elastic Beanstalk environments, and will not be terminated or monitored by Elastic Beanstalk.

In the following steps you'll use the Amazon RDS console to:

- Launch a database with the **MySQL** engine.
- Enable a **Multi-AZ deployment**. This creates a standby in a different Availability Zone (AZ) to provide data redundancy, eliminate I/O freezes, and minimize latency spikes during system backups.

To launch an RDS DB instance in a default VPC

1. Open the RDS console.
2. In the navigation pane, choose **Databases**.
3. Choose **Create database**.
4. Choose **Standard Create**.

   Important

   Do not choose **Easy Create**. If you choose it, you can't configure the necessary settings to launch this RDS DB.

5. Under **Additional configuration**, for **Initial database name**, type **ebdb**.
6. Review the default settings and adjust these settings according to your specific requirements. Pay attention to the following options:
   - **DB instance class** – Choose an instance size that has an appropriate amount of memory and CPU power for your workload.
   - **Multi-AZ deployment** – For high availability, set this to **Create an Aurora Replica/Reader node in a different AZ**.
   - **Master username** and **Master password** – The database username and password. Make a note of these settings because you will use them later.
7. Verify the default settings for the remaining options, and then choose **Create database**.

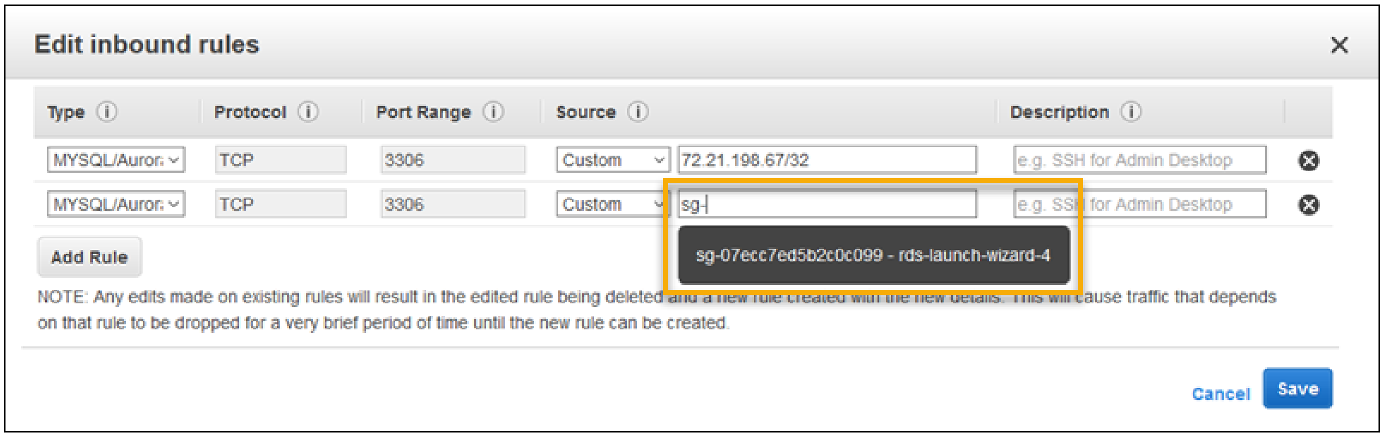
After your DB instance is created, modify the security group attached to it in order to allow inbound traffic on the appropriate port..

Note

This is the same security group that you'll attach to your Elastic Beanstalk environment later, so the rule that you add now will grant ingress permission to other resources in the same security group.

To modify the inbound rules on the security group that's attached to your RDS instance

1. Open the [Amazon RDS console](#).
2. Choose **Databases**.
3. Choose the name of your DB instance to view its details.
4. In the **Connectivity** section, make a note of the **Subnets**, **Security groups**, and **Endpoint** that are displayed on this page. This is so you can use this information later.
5. Under **Security**, you can see the security group that's associated with the DB instance. Open the link to view the security group in the Amazon EC2 console.
6. In the security group details, choose **Inbound**.
7. Choose **Edit**.
8. Choose **Add Rule**.
9. For **Type**, choose the DB engine that your application uses.
10. For **Source**, type **sg-** to view a list of available security groups. Choose the security group that's associated with the Auto Scaling group that's used with your Elastic Beanstalk environment. This is so that Amazon EC2 instances in the environment can have access to the database.



Edit inbound rules

Type	Protocol	Port Range	Source		Description
MYSQL/Auror:	TCP	3306	Custom	72.21.198.67/32	e.g. SSH for Admin Desktop
MYSQL/Auror:	TCP	3306	Custom	sg-	e.g. SSH for Admin Desktop

sg-07ecc7ed5b2c0c099 - rds-launch-wizard-4

NOTE: Any edits made on existing rules will result in the edited rule being deleted and a new rule created with the new details. This will cause traffic that depends on that rule to be dropped for a very brief period of time until the new rule can be created.

11. Choose **Save**.

Creating a DB instance takes about 10 minutes. In the meantime, download WordPress and create your Elastic Beanstalk environment.

Download WordPress

To prepare to deploy WordPress using AWS Elastic Beanstalk, you must copy the WordPress files to your computer and provide the correct configuration information.

To create a WordPress project

1. Download WordPress from wordpress.org.

```
~$curl https://wordpress.org/wordpress-6.2.tar.gz -o wordpress.tar.gz
```

2. Download the configuration files from the sample repository.

```
~$ wget https://github.com/aws-samples/eb-php-wordpress/releases/download/v1.1/eb-php-wordpress-v1.zip
```

3. Extract WordPress and change the name of the folder.

```
~$ tar -xvf wordpress.tar.gz
~$ mv wordpress wordpress-beanstalk
~$ cd wordpress-beanstalk
```

4. Extract the configuration files over the WordPress installation.

```
~/wordpress-beanstalk$ unzip ../eb-php-wordpress-v1.zip
 creating: .ebextensions/
inflating: .ebextensions/dev.config
inflating: .ebextensions/efs-create.config
inflating: .ebextensions/efs-mount.config
inflating: .ebextensions/loadbalancer-sg.config
inflating: .ebextensions/wordpress.config
inflating: LICENSE
inflating: README.md
inflating: wp-config.php
```

Launch an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. After you launch the environment, you can configure it to connect to the database, then deploy the WordPress code to the environment.

In the following steps, you'll use the Elastic Beanstalk console to:

- Create an Elastic Beanstalk application using the managed **PHP** platform.
- Accept the default settings and sample code.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link: console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about five minutes and creates the following resources.

Elastic Beanstalk created resources

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

  Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or NGINX as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.

- **Load balancer –** An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group –** An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group –** An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket –** A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms –** Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack –** Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name –** A domain name that routes to your web app in the form *`subdomain`*.*`region`*.*elasticbeanstalk.com*.

Domain security

To augment the security of your Elastic Beanstalk applications, the *elasticbeanstalk.com* domain is registered in the Public Suffix List (PSL).
If you ever need to set sensitive cookies in the default domain name for your Elastic Beanstalk applications, we recommend that you use cookies with a `__Host-` prefix for increased security. This practice defends your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Because the Amazon RDS instance that you launched is outside of your environment, you are responsible for managing its lifecycle.

Note

The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see Using Elastic Beanstalk with Amazon S3.

Configure security groups and environment properties

Add the security group of your DB instance to your running environment. This procedure causes Elastic Beanstalk to reprovision all instances in your environment with the additional security group attached.

To add a security group to your environment

- Do one of the following:
  - To add a security group using the Elastic Beanstalk console
    a. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
    b. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
    c. In the navigation pane, choose **Configuration**.
    d. In the **Instances** configuration category, choose **Edit**.
    e. Under **EC2 security groups**, choose the security group to attach to the instances, in addition to the instance security group that Elastic Beanstalk creates.
    f. To save the changes choose **Apply** at the bottom of the page.
    g. Read the warning, and then choose **Confirm**.
  - To add a security group using a configuration file, use the `securitygroup-addexisting.config` example file.

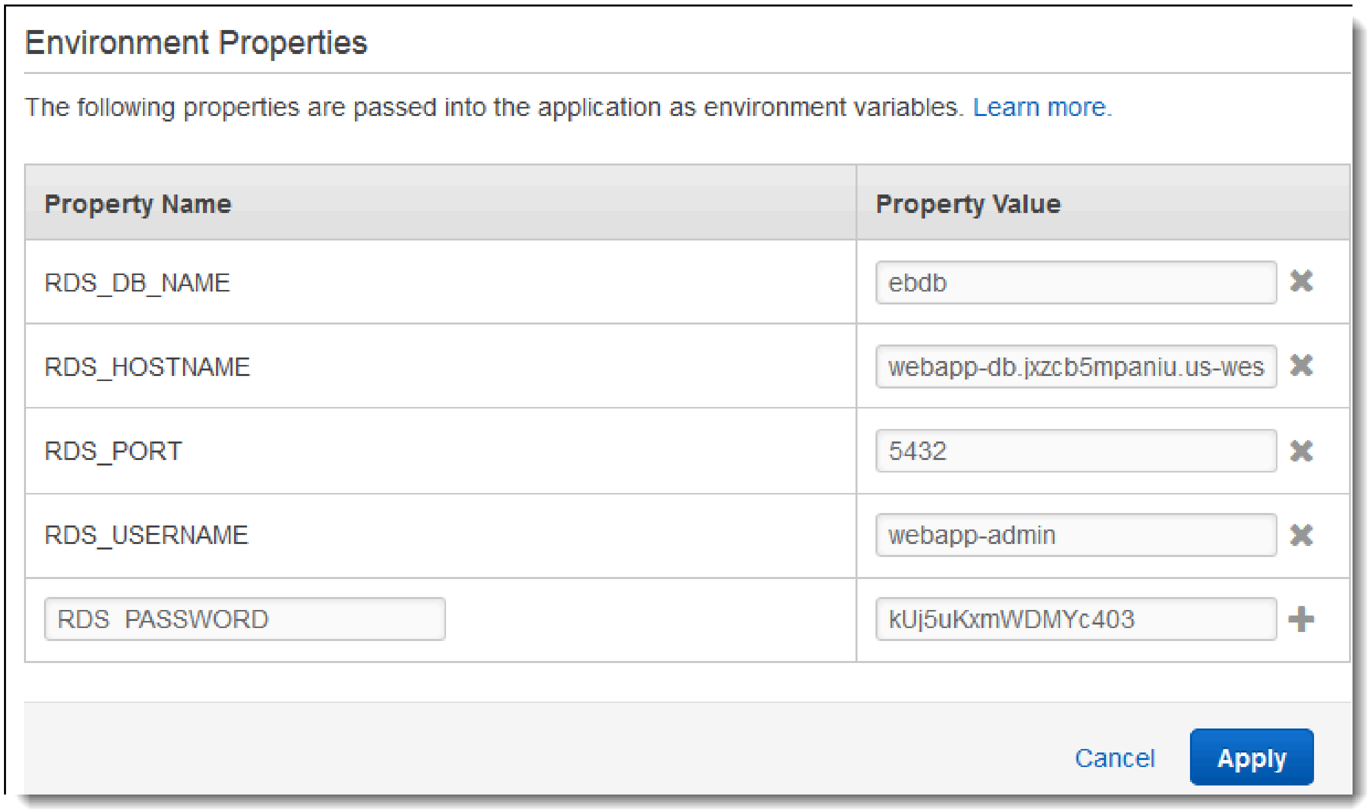
Next, use environment properties to pass the connection information to your environment.

The WordPress application uses a default set of properties that match the ones that Elastic Beanstalk configures when you provision a database within your environment.

To configure environment properties for an Amazon RDS DB instance

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. In the **Environment properties** section, define the variables that your application reads to construct a connection string. For compatibility with environments that have an integrated RDS DB instance, use the following names and values. You can find all values, except for your password, in the RDS console.

Property name	Description	Property value
`RDS_HOSTNAME`	The hostname of the DB instance.	On the **Connectivity & security** tab on the Amazon RDS console: **Endpoint**.
`RDS_PORT`	The port where the DB instance accepts connectio ns. The default value varies among DB engines.	On the **Connectivity & security** tab on the Amazon RDS console: **Port**.
`RDS_DB_NAME`	The database name, **ebdb**.	On the **Configuration** tab on the Amazon RDS console: **DB Name**.
`RDS_USERNAME`	The username that you configured for your database.	On the **Configuration** tab on the Amazon RDS console: **Master username**.
`RDS_PASSWORD`	The password that you configured for your database.	Not available for reference in the Amazon RDS console.



Environment Properties

The following properties are passed into the application as environment variables. Learn more.

Property Name	Property Value
RDS_DB_NAME	ebdb
RDS_HOSTNAME	webapp-db.jxzcb5mpaniu.us-wes
RDS_PORT	5432
RDS_USERNAME	webapp-admin
RDS_PASSWORD	kUj5uKxmWDMYc403

6. To save the changes choose **Apply** at the bottom of the page.

Configure and deploy your application

Verify that the structure of your `wordpress-beanstalk` folder is correct, as shown.

```
wordpress-beanstalk$ tree -aL 1
.
### .ebextensions
### index.php
### LICENSE
### license.txt
### readme.html
### README.md
### wp-activate.php
### wp-admin
### wp-blog-header.php
### wp-comments-post.php
### wp-config.php
### wp-config-sample.php
### wp-content
### wp-cron.php
### wp-includes
### wp-links-opml.php
### wp-load.php
### wp-login.php
### wp-mail.php
### wp-settings.php
### wp-signup.php
### wp-trackback.php
### xmlrpc.php
```

The customized `wp-config.php` file from the project repo uses the environment variables that you defined in the previous step to configure the database connection. The `.ebextensions` folder contains configuration files that create additional resources within your Elastic Beanstalk environment.

The configuration files require modification to work with your account. Replace the placeholder values in the files with the appropriate IDs and create a source bundle.

To update configuration files and create a source bundle

1. Modify the configuration files as follows.

   - `.ebextensions/dev.config` – Restricts access to your environment to protect it during the WordPress installation process. Replace the placeholder IP address near the top of the file with the public IP address of the computer you'll use to access your environment's website to complete your WordPress installation.

     Note

     Depending on your network, you might need to use an IP address block.

   - `.ebextensions/efs-create.config` – Creates an EFS file system and mount points in each Availability Zone/subnet in your VPC. Identify your default VPC and subnet IDs in the Amazon VPC console.

2. Create a source bundle containing the files in your project folder. The following command creates a source bundle named `wordpress-beanstalk.zip`.

   ```
   ~/eb-wordpress$ zip ../wordpress-beanstalk.zip -r * .[^.]*
   ```

Upload the source bundle to Elastic Beanstalk to deploy WordPress to your environment.

To deploy a source bundle

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Install WordPress

To complete your WordPress installation

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose the environment URL to open your site in a browser. You are redirected to a WordPress installation wizard because you haven't configured the site yet.
4. Perform a standard installation. The `wp-config.php` file is already present in the source code and configured to read the database connection information from the environment. You shouldn't be prompted to configure the connection.

Installation takes about a minute to complete.

Update keys and salts

The WordPress configuration file `wp-config.php` also reads values for keys and salts from environment properties. Currently, these properties are all set to `test` by the `wordpress.config` file in the `.ebextensions` folder.

The hash salt can be any value that meets the environment property requirements, but you should not store it in source control. Use the Elastic Beanstalk console to set these properties directly on the environment.

To update environment properties

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the navigation pane, choose **Configuration**.
4. Under **Software**, choose **Edit**.
5. For `Environment properties`, modify the following properties:
   - `AUTH_KEY` – The value chosen for `AUTH_KEY`.
   - `SECURE_AUTH_KEY` – The value chosen for `SECURE_AUTH_KEY`.
   - `LOGGED_IN_KEY` – The value chosen for `LOGGED_IN_KEY`.
   - `NONCE_KEY` – The value chosen for `NONCE_KEY`.
   - `AUTH_SALT` – The value chosen for `AUTH_SALT`.
   - `SECURE_AUTH_SALT` – The value chosen for `SECURE_AUTH_SALT`.
   - `LOGGED_IN_SALT` – The value chosen for `LOGGED_IN_SALT`.
   - `NONCE_SALT` — The value chosen for `NONCE_SALT`.
6. To save the changes choose **Apply** at the bottom of the page.

Note

Setting the properties on the environment directly overrides the values in `wordpress.config`.

Remove access restrictions

The sample project includes the configuration file `loadbalancer-sg.config`. It creates a security group and assigns it to the environment's load balancer, using the IP address that you configured in `dev.config`. It restricts HTTP access on port 80 to connections from your network. Otherwise, an outside party could potentially connect to your site before you have installed WordPress and configured your admin account.

Now that you've installed WordPress, remove the configuration file to open the site to the world.

To remove the restriction and update your environment

1. Delete the `.ebextensions/loadbalancer-sg.config` file from your project directory.

   ```
   ~/wordpress-beanstalk$ rm .ebextensions/loadbalancer-sg.config
   ```

2. Create a source bundle.

   ```
   ~/eb-wordpress$ zip ../wordpress-beanstalk-v2.zip -r * .[^.]*
   ```

Upload the source bundle to Elastic Beanstalk to deploy WordPress to your environment.

To deploy a source bundle

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Configure your Auto Scaling group

Finally, configure your environment's Auto Scaling group with a higher minimum instance count. Run at least two instances at all times to prevent the web servers in your environment from being a single point of failure. This also allows you to deploy changes without taking your site out of service.

To configure your environment's Auto Scaling group for high availability

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.

4. In the **Capacity** configuration category, choose **Edit**.
5. In the **Auto Scaling group** section, set **Min instances** to **2**.
6. To save the changes choose **Apply** at the bottom of the page.

To support content uploads across multiple instances, the sample project uses Amazon EFS to create a shared file system. Create a post on the site and upload content to store it on the shared file system. View the post and refresh the page multiple times to hit both instances and verify that the shared file system is working.

Upgrade WordPress

To upgrade to a new version of WordPress, back up your site and deploy it to a new environment.

Important

Do not use the update functionality within WordPress or update your source files to use a new version. Both of these actions can result in your post URLs returning 404 errors even though they are still in the database and file system.

To upgrade WordPress

1. In the WordPress admin console, use the export tool to export your posts to an XML file.
2. Deploy and install the new version of WordPress to Elastic Beanstalk with the same steps that you used to install the previous version. To avoid downtime, you can create an environment with the new version.
3. On the new version, install the WordPress Importer tool in the admin console and use it to import the XML file containing your posts. If the posts were created by the admin user on the old version, assign them to the admin user on the new site instead of trying to import the admin user.
4. If you deployed the new version to a separate environment, do a CNAME swap to redirect users from the old site to the new site.

Clean up

After you finish working with the demo code, you can terminate your environment. Elastic Beanstalk deletes all related AWS resources, such as Amazon EC2 instances, database instances, load balancers, security groups, and alarms.

Removing resources does not delete the Elastic Beanstalk application, so you can create new environments for your application at any time.

To terminate your Elastic Beanstalk environment from the console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

In addition, you can terminate database resources that you created outside of your Elastic Beanstalk environment. When you terminate an Amazon RDS DB instance, you can take a snapshot and restore the data to another instance later.

To terminate your RDS DB instance

1. Open the Amazon RDS console.
2. Choose **Databases**.
3. Choose your DB instance.
4. Choose **Actions**, and then choose **Delete**.
5. Choose whether to create a snapshot, and then choose **Delete**.

Next steps

As you continue to develop your application, you'll probably want a way to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The Elastic Beanstalk Command Line Interface (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

The sample application uses configuration files to configure PHP settings and create a table in the database, if it doesn't already exist. You can also use a configuration file to configure the security group settings of your instances during environment creation to avoid time-consuming configuration updates. See Advanced environment customization with configuration files (`.ebextensions`) for more information.

For development and testing, you might want to use the Elastic Beanstalk functionality for adding a managed DB instance directly to your environment. For instructions on setting up a database inside your environment, see Adding a database to your Elastic Beanstalk environment.

If you need a high-performance database, consider using Amazon Aurora. Amazon Aurora is a MySQL-compatible database engine that offers commercial database features at low cost. To connect your application to a different database, repeat the security group configuration steps and update the RDS-related environment properties.

Finally, if you plan on using your application in a production environment, you will want to configure a custom domain name for your environment and enable HTTPS for secure connections.

Deploying a high-availability Drupal website with an external Amazon RDS database to Elastic Beanstalk

This tutorial walks you through the process of launching an RDS DB instance external to AWS Elastic Beanstalk. Then it describes configuring a high-availability environment running a Drupal website to connect to it. The website uses Amazon Elastic File System (Amazon EFS) as shared storage for uploaded files. Running a DB instance external to Elastic Beanstalk decouples the database from the lifecycle of your environment, and lets you connect to the same database from multiple environments, swap out one database for another, or perform a blue/green deployment without affecting your database.

Sections

Prerequisites

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in Learn how to get started with Elastic Beanstalk to launch your first Elastic Beanstalk environment.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

The procedures in this tutorial for Amazon Relational Database Service (Amazon RDS) tasks assume that you are launching resources in a default Amazon Virtual Private Cloud (Amazon VPC). All new accounts include a default VPC in each region. If you don't have a default VPC, the procedures will vary. See Using Elastic Beanstalk with Amazon RDS for instructions for EC2-Classic and custom VPC platforms.

The sample application uses Amazon EFS. It only works in AWS Regions that support Amazon EFS. To learn about supporting AWS Regions, see Amazon Elastic File System Endpoints and Quotas in the *AWS General Reference*.

If the platform of your Elastic Beanstalk environment uses PHP 7.4 or earlier, we recommend that you use Drupal version 8.9.13 for this tutorial. For platforms installed with PHP 8.0 or later, we recommend that you use Drupal 9.1.5.

For more information about Drupal releases and the PHP versions that they support, see PHP requirements on the Drupal website. The core versions that Drupal recommends are listed on the website https://www.drupal.org/project/drupal.

Launch a DB instance in Amazon RDS

To use an external database with an application running in Elastic Beanstalk, first launch a DB instance with Amazon RDS. When you launch an instance with Amazon RDS, it is completely independent of Elastic Beanstalk and your Elastic Beanstalk environments, and will not be terminated or monitored by Elastic Beanstalk.

Use the Amazon RDS console to launch a Multi-AZ **MySQL** DB instance. Choosing a Multi-AZ deployment ensures that your database will failover and continue to be available if the source DB instance goes out of service.

To launch an RDS DB instance in a default VPC

1. Open the RDS console.
2. In the navigation pane, choose **Databases**.
3. Choose **Create database**.
4. Choose **Standard Create**.

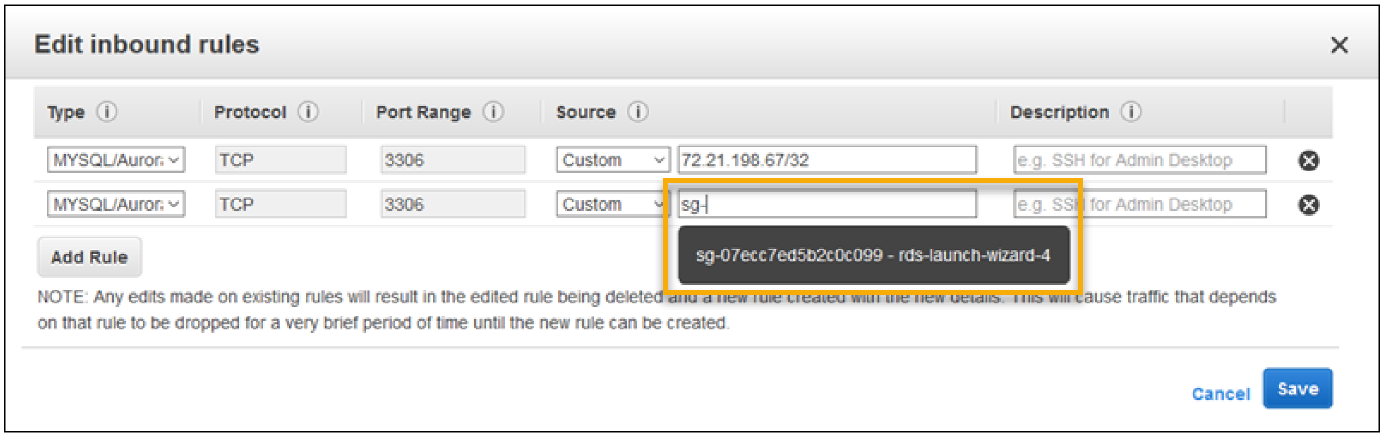
> ⚠ **Important**
>
> Do not choose **Easy Create**. If you choose it, you can't configure the necessary settings to launch this RDS DB.

5. Under **Additional configuration**, for **Initial database name**, type **ebdb**.
6. Review the default settings and adjust these settings according to your specific requirements. Pay attention to the following options:
   - **DB instance class** – Choose an instance size that has an appropriate amount of memory and CPU power for your workload.
   - **Multi-AZ deployment** – For high availability, set this to **Create an Aurora Replica/Reader node in a different AZ**.
   - **Master username** and **Master password** – The database username and password. Make a note of these settings because you will use them later.
7. Verify the default settings for the remaining options, and then choose **Create database**.

Next, modify the security group attached to your DB instance to allow inbound traffic on the appropriate port. This is the same security group that you will attach to your Elastic Beanstalk environment later, so the rule that you add will grant ingress permission to other resources in the same security group.

To modify the inbound rules on the security group that's attached to your RDS instance

1. Open the [Amazon RDS console](https://console.aws.amazon.com/rds/).
2. Choose **Databases**.
3. Choose the name of your DB instance to view its details.
4. In the **Connectivity** section, make a note of the **Subnets**, **Security groups**, and **Endpoint** that are displayed on this page. This is so you can use this information later.
5. Under **Security**, you can see the security group that's associated with the DB instance. Open the link to view the security group in the Amazon EC2 console.
6. In the security group details, choose **Inbound**.
7. Choose **Edit**.
8. Choose **Add Rule**.
9. For **Type**, choose the DB engine that your application uses.
10. For **Source**, type **sg-** to view a list of available security groups. Choose the security group that's associated with the Auto Scaling group that's used with your Elastic Beanstalk environment. This is so that Amazon EC2 instances in the environment can have access to the database.



Edit inbound rules

Type	Protocol	Port Range	Source		Description
MYSQL/Auror:	TCP	3306	Custom	72.21.198.67/32	e.g. SSH for Admin Desktop
MYSQL/Auror:	TCP	3306	Custom	sg-	e.g. SSH for Admin Desktop

sg-07ecc7ed5b2c0c099 - rds-launch-wizard-4

NOTE: Any edits made on existing rules will result in the edited rule being deleted and a new rule created with the new details. This will cause traffic that depends on that rule to be dropped for a very brief period of time until the new rule can be created.

11. Choose **Save**.

Creating a DB instance takes about 10 minutes. In the meantime, launch your Elastic Beanstalk environment.

Launch an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. Choose the **PHP** platform and accept the default settings and sample code. After you launch the environment, you can configure the environment to connect to the database, then deploy the Drupal code to the environment.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link: console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about 5 minutes and creates the following resources:

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

  Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or NGINX as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.

- **Load balancer security group –** An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group –** An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket –** A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms –** Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack –** Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name –** A domain name that routes to your web app in the form *`subdomain`.`region`.elasticbeanstalk.com*.

> **Domain security**
>
> To augment the security of your Elastic Beanstalk applications, the *elasticbeanstalk.com* domain is registered in the Public Suffix List (PSL).
> If you ever need to set sensitive cookies in the default domain name for your Elastic Beanstalk applications, we recommend that you use cookies with a `__Host-` prefix for increased security. This practice defends your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains. The RDS DB instance that you launched is outside of your environment, so you are responsible for managing its lifecycle.

> **Note**
>
> The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see Using Elastic Beanstalk with Amazon S3.

Configure security settings and environment properties

Add the security group of your DB instance to your running environment. This procedure causes Elastic Beanstalk to reprovision all instances in your environment with the additional security group attached.

To add a security group to your environment

- Do one of the following:
  - To add a security group using the Elastic Beanstalk console
    a. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
    b. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
    c. In the navigation pane, choose **Configuration**.
    d. In the **Instances** configuration category, choose **Edit**.
    e. Under **EC2 security groups**, choose the security group to attach to the instances, in addition to the instance security group that Elastic Beanstalk creates.
    f. To save the changes choose **Apply** at the bottom of the page.
    g. Read the warning, and then choose **Confirm**.
  - To add a security group using a configuration file, use the `securitygroup-addexisting.config` example file.

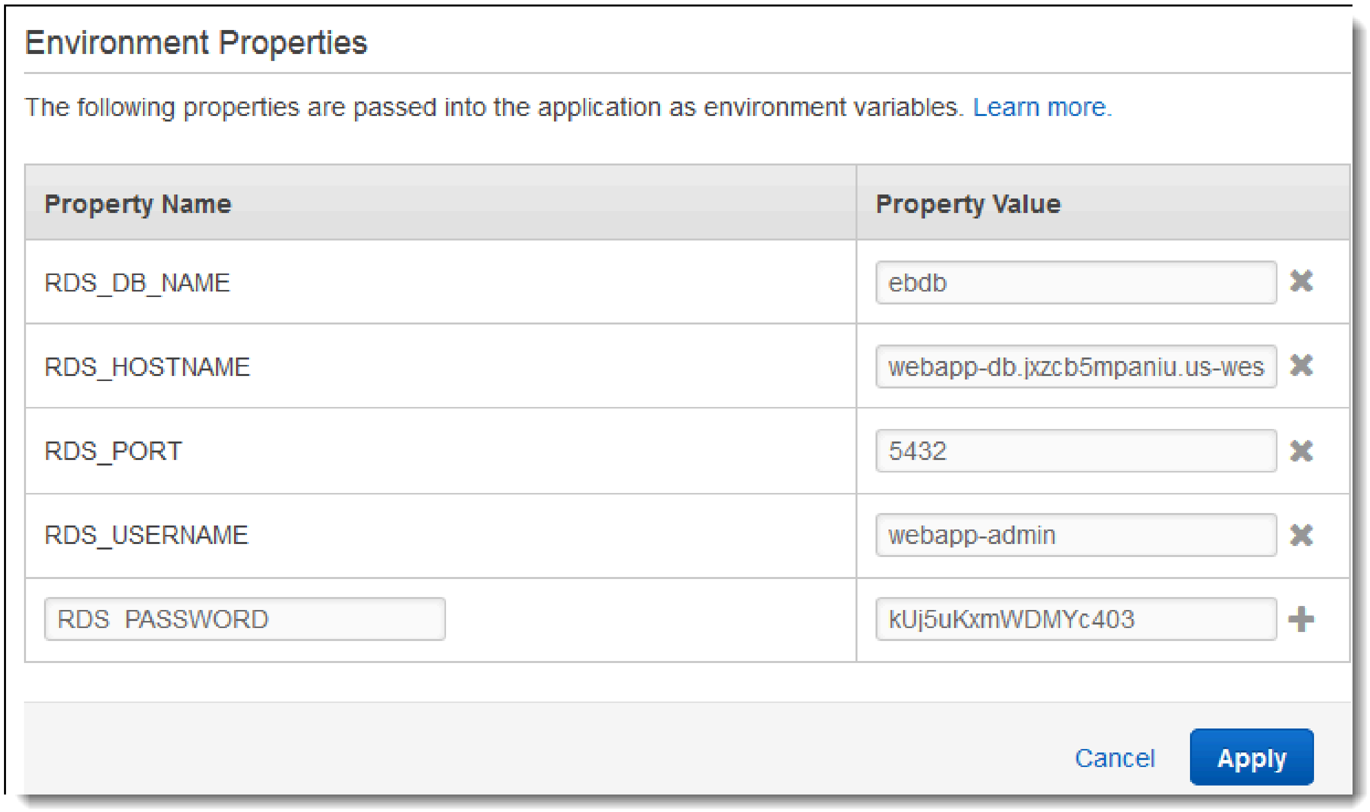
Next, use environment properties to pass the connection information to your environment. The sample application uses a default set of properties that match the ones that Elastic Beanstalk configures when you provision a database within your environment.

To configure environment properties for an Amazon RDS DB instance

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. In the **Environment properties** section, define the variables that your application reads to construct a connection string. For compatibility with environments that have an integrated

RDS DB instance, use the following names and values. You can find all values, except for your password, in the RDS console.

Property name	Description	Property value
`RDS_HOSTNAME`	The hostname of the DB instance.	On the **Connectivity & security** tab on the Amazon RDS console: **Endpoint**.
`RDS_PORT`	The port where the DB instance accepts connectio ns. The default value varies among DB engines.	On the **Connectivity & security** tab on the Amazon RDS console: **Port**.
`RDS_DB_NAME`	The database name, **ebdb**.	On the **Configuration** tab on the Amazon RDS console: **DB Name**.
`RDS_USERNAME`	The username that you configured for your database.	On the **Configuration** tab on the Amazon RDS console: **Master username**.
`RDS_PASSWORD`	The password that you configured for your database.	Not available for reference in the Amazon RDS console.


Environment Properties

The following properties are passed into the application as environment variables. Learn more.

Property Name	Property Value
RDS_DB_NAME	ebdb
RDS_HOSTNAME	webapp-db.jxzcb5mpaniu.us-wes
RDS_PORT	5432
RDS_USERNAME	webapp-admin
RDS_PASSWORD	kUj5uKxmWDMYc403

6. To save the changes choose **Apply** at the bottom of the page.

After installing Drupal, you need to connect to the instance with SSH to retrieve some configuration details. Assign an SSH key to your environment's instances.

To configure SSH

1. If you haven't previously created a key pair, open the key pairs page of the Amazon EC2 console and follow the instructions to create one.
2. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
3. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
4. In the navigation pane, choose **Configuration**.
5. Under **Security**, choose **Edit**.
6. For **EC2 key pair**, choose your key pair.
7. To save the changes choose **Apply** at the bottom of the page.

Configure and deploy your application

To create a Drupal project for Elastic Beanstalk, download the Drupal source code and combine it with the files in the aws-samples/eb-php-drupal repository on GitHub.

To create a Drupal project

1. Run the follwing command to download Drupal from *www.drupal.org/download*. To learn more about downloads, see the the Drupal website.

   If the platform of your Elastic Beanstalk environment uses PHP 7.4 or earlier, we recommend that you download Drupal version 8.9.13 for this tutorial. You can run the following command to download it.

   ```
   ~$ curl https://ftp.drupal.org/files/projects/drupal-8.9.13.tar.gz -o drupal.tar.gz
   ```

   If your platform uses PHP 8.0 or later, we recommend that you download Drupal 9.1.5. You can use this command to download it.

   ```
   ~$ curl https://ftp.drupal.org/files/projects/drupal-9.1.5.tar.gz -o drupal.tar.gz
   ```

   For more information about Drupal releases and the PHP versions that they support, see PHP requirements in the official Drupal documentation. The core versions that Drupal recommends are listed on the Drupal website.

2. Use the following command to download the configuration files from the sample repository:

   ```
   ~$ wget https://github.com/aws-samples/eb-php-drupal/releases/download/v1.1/eb-php-drupal-v1.zip
   ```

3. Extract Drupal and change the name of the folder.

   If you downloaded Drupal 8.9.13:

   ```
    ~$ tar -xvf drupal.tar.gz
    ~$ mv drupal-8.9.13 drupal-beanstalk
    ~$ cd drupal-beanstalk
   ```

   If you downloaded Drupal 9.1.5:

```
~$ tar -xvf drupal.tar.gz
~$ mv drupal-9.1.5 drupal-beanstalk
~$ cd drupal-beanstalk
```

4. Extract the configuration files over the Drupal installation.

```
~/drupal-beanstalk$ unzip ../eb-php-drupal-v1.zip
 creating: .ebextensions/
 inflating: .ebextensions/dev.config
 inflating: .ebextensions/drupal.config
 inflating: .ebextensions/efs-create.config
 inflating: .ebextensions/efs-filesystem.template
 inflating: .ebextensions/efs-mount.config
 inflating: .ebextensions/loadbalancer-sg.config
 inflating: LICENSE
 inflating: README.md
 inflating: beanstalk-settings.php
```

Verify that the structure of your `drupal-beanstalk` folder is correct, as shown.

```
drupal-beanstalk$ tree -aL 1
.
### autoload.php
### beanstalk-settings.php
### composer.json
### composer.lock
### core
### .csslintrc
### .ebextensions
### .ebextensions
### .editorconfig
### .eslintignore
### .eslintrc.json
### example.gitignore
### .gitattributes
### .htaccess
### .ht.router.php
### index.php
### LICENSE
### LICENSE.txt
### modules
### profiles
### README.md
### README.txt
### robots.txt
### sites
### themes
### update.php
### vendor
### web.config
```

The `beanstalk-settings.php` file from the project repo uses the environment variables that you defined in the previous step to configure the database connection. The `.ebextensions` folder contains configuration files that create additional resources within your Elastic Beanstalk environment.

The configuration files require modification to work with your account. Replace the placeholder values in the files with the appropriate IDs and create a source bundle.

To update configuration files and create a source bundle.

1. Modify the configuration files as follows.

   - `.ebextensions/dev.config` – restricts access to your environment to your IP address to protect it during the Drupal installation process. Replace the placeholder IP address near the top of the file with your public IP address.
   - `.ebextensions/efs-create.config` – creates an EFS file system and mount points in each Availability Zone / subnet in your VPC. Identify your default VPC and subnet IDs in the Amazon VPC console.

2. Create a source bundle containing the files in your project folder. The following command creates a source bundle named `drupal-beanstalk.zip`. It excludes files in the `vendor` folder, which take up a lot of space and are not necessary for deploying your application to Elastic Beanstalk.

   ```
   ~/eb-drupal$ zip ../drupal-beanstalk.zip -r * .[^.]* -x "vendor/*"
   ```

Upload the source bundle to Elastic Beanstalk to deploy Drupal to your environment.

To deploy a source bundle

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Install Drupal

To complete your Drupal installation

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose the environment URL to open your site in a browser. You are redirected to a Drupal installation wizard because the site has not been configured yet.
4. Perform a standard installation with the following settings for the database:
    - **Database name** – The **DB Name** shown in the Amazon RDS console.
    - **Database username and password** – The **Master Username** and **Master Password** values you entered when creating your database.
    - **Advanced Options > Host** – The **Endpoint** of the DB instance shown in the Amazon RDS console.

Installation takes about a minute to complete.

Update Drupal configuration and remove access restrictions

The Drupal installation process created a file named `settings.php` in the `sites/default` folder on the instance. You need this file in your source code to avoid resetting your site on subsequent deployments, but the file currently contains secrets that you don't want to commit to source. Connect to the application instance to retrieve information from the settings file.

To connect to your application instance with SSH

1. Open the instances page of the Amazon EC2 console.
2. Choose the application instance. It is the one named after your Elastic Beanstalk environment.
3. Choose **Connect**.
4. Follow the instructions to connect the instance with SSH. The command looks similar to the following.

   ```
   $ ssh -i ~/.ssh/mykey ec2-user@ec2-00-55-33-222.us-west-2.compute.amazonaws.com
   ```

Get the sync directory id from the last line of the settings file.

```
[ec2-user ~]$ tail -n 1 /var/app/current/sites/default/settings.php
$config_directories['sync'] = 'sites/default/files/
config_4ccfX2sPQm79p1mk5IbUq9S_FokcENO4mxyC-L18-4g_xKj_7j9ydn31kDOYOgnzMu071Tvc4Q/
sync';
```

The file also contains the sites current hash key, but you can ignore the current value and use your own.

Assign the sync directory path and hash key to environment properties. The customized settings file from the project repo reads these properties to configure the site during deployment, in addition to the database connection properties that you set earlier.

Drupal configuration properties

- SYNC_DIR – The path to the sync directory.
- HASH_SALT – Any string value that meets environment property requirements.

To configure environment variables in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.

5. Scroll down to **Runtime environment variables**.
6. Select **Add environment variable**.
7. For **Source** select **Plain text**.

   Note

   The **Secrets Manager** and **SSM Parameter Store** values in the drop-down are for configuring environment variables as secrets to store sensitive data, such as credentials and API keys. For more information, see Using Elastic Beanstalk with AWS Secrets Manager and AWS Systems Manager Parameter Store.

8. Enter the **Environment variable name** and **Environment variable value** pairs.
9. If you need to add more variables repeat **Step 6** through **Step 8**.
10. To save the changes choose **Apply** at the bottom of the page.

Finally, the sample project includes a configuration file (`loadbalancer-sg.config`) that creates a security group and assigns it to the environment's load balancer, using the IP address that you configured in `dev.config` to restrict HTTP access on port 80 to connections from your network. Otherwise, an outside party could potentially connect to your site before you have installed Drupal and configured your admin account.

To update Drupal's configuration and remove access restrictions

1. Delete the `.ebextensions/loadbalancer-sg.config` file from your project directory.

```
~/drupal-beanstalk$ rm .ebextensions/loadbalancer-sg.config
```

2. Copy the customized `settings`.php file into the sites folder.

```
~/drupal-beanstalk$ cp beanstalk-settings.php sites/default/settings.php
```

3. Create a source bundle.

```
~/eb-drupal$ zip ../drupal-beanstalk-v2.zip -r * .[^.]* -x "vendor/*"
```

Upload the source bundle to Elastic Beanstalk to deploy Drupal to your environment.

To deploy a source bundle

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Configure your Auto Scaling group

Finally, configure your environment's Auto Scaling group with a higher minimum instance count. Run at least two instances at all times to prevent the web servers in your environment from being a single point of failure, and to allow you to deploy changes without taking your site out of service.

To configure your environment's Auto Scaling group for high availability

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Capacity** configuration category, choose **Edit**.
5. In the **Auto Scaling group** section, set **Min instances** to **2**.
6. To save the changes choose **Apply** at the bottom of the page.

To support content uploads across multiple instances, the sample project uses Amazon Elastic File System to create a shared file system. Create a post on the site and upload content to store it on the shared file system. View the post and refresh the page multiple times to hit both instances and verify that the shared file system is working.

Cleanup

After you finish working with the demo code, you can terminate your environment. Elastic Beanstalk deletes all related AWS resources, such as Amazon EC2 instances, database instances, load balancers, security groups, and alarms.

Removing resources does not delete the Elastic Beanstalk application, so you can create new environments for your application at any time.

To terminate your Elastic Beanstalk environment from the console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

In addition, you can terminate database resources that you created outside of your Elastic Beanstalk environment. When you terminate an Amazon RDS DB instance, you can take a snapshot and restore the data to another instance later.

To terminate your RDS DB instance

1. Open the Amazon RDS console.
2. Choose **Databases**.
3. Choose your DB instance.
4. Choose **Actions**, and then choose **Delete**.
5. Choose whether to create a snapshot, and then choose **Delete**.

Next steps

As you continue to develop your application, you'll probably want a way to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The Elastic Beanstalk Command Line Interface (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

The sample application uses configuration files to configure PHP settings and create a table in the database if it doesn't already exist. You can also use a configuration file to configure your instances' security group settings during environment creation to avoid time-consuming configuration updates. See Advanced environment customization with configuration files (`.ebextensions`) for more information.

For development and testing, you might want to use the Elastic Beanstalk functionality for adding a managed DB instance directly to your environment. For instructions on setting up a database inside your environment, see Adding a database to your Elastic Beanstalk environment.

If you need a high-performance database, consider using Amazon Aurora. Amazon Aurora is a MySQL-compatible database engine that offers commercial database features at low cost. To connect your application to a different database, repeat the security group configuration steps and update the RDS-related environment properties.

Finally, if you plan on using your application in a production environment, you will want to configure a custom domain name for your environment and enable HTTPS for secure connections.

Deploying Python applications with Elastic Beanstalk

This chapter provides instructions for configuring and deploying your Python web application to AWS Elastic Beanstalk. Elastic Beanstalk makes it easy to deploy, manage, and scale your Python web applications using Amazon Web Services.

You can deploy your application in just a few minutes using the Elastic Beanstalk Command Line Interface (EB CLI) or by using the Elastic Beanstalk console. After you deploy your Elastic Beanstalk application, you can continue to use the EB CLI to manage your application and environment, or you can use the Elastic Beanstalk console, AWS CLI, or the APIs.

Follow the steps in the QuickStart for Python for step-by-step instructions to create and deploy a Python *Hello World* web application with the EB CLI.

Topics

QuickStart: Deploy a Python application to Elastic Beanstalk

This QuickStart tutorial walks you through the process of creating a Python application and deploying it to an AWS Elastic Beanstalk environment.

Note

Tutorial examples are intended for demonstration. Do not use the application for production traffic.

Sections

Your AWS account

If you're not already an AWS customer, you need to create an AWS account. Signing up enables you to access Elastic Beanstalk and other AWS services that you need.

If you already have an AWS account, you can move on to Prerequisites.

Create an AWS account

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Prerequisites

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

EB CLI

This tutorial uses the Elastic Beanstalk Command Line Interface (EB CLI). For details on installing and configuring the EB CLI, see Install EB CLI with setup script (recommended) and Configure the EB CLI.

Python and Flask framework

Confirm that you have a working Python version with `pip` installed by running the following commands.

```
~$ python3 --version
Python 3.N.N
```

```
>~$ python3 -m pip --version
pip X.Y.Z from ... (python 3.N.N)
```

If any of the previous commands return "*Python was not found*", run the following commands that use `python` instead of `python3`. The setup of aliases and symbolic links can vary by operating system and individual customizations, so the `python3` command may not function on your machine.

```
~$ python --version
Python 3.N.N
>~$ python -m pip --version
pip X.Y.Z from ... (python 3.N.N)
```

If you don't have Python installed on your local machine, you can download it from the Python downloads page on the Python website. For a list of Python language versions supported by Elastic Beanstalk, see Supported Python platforms in the *AWS Elastic Beanstalk Platforms* guide. The Python downloads website provides a link to the *Python Developer's Guide*, where you'll find installation and setup instructions.

Note

The Python `pip` package is included by default with Python 3.4 or later.

If your output indicates that you have a supported version of Python, but not `pip`, see the Installation page on the *pip.pypa.io* website. It provides guidance to install pip within a Python environment that doesn't have it.

Confirm if Flask is installed by running the following command:

```
~$ pip list | grep Flask
```

If Flask is not installed, you can install it with the following command:

```
~$ pip install Flask
```

Step 1: Create a Python application

Create a project directory.

```
~$ mkdir eb-python
~$ cd eb-python
```

Create a sample "Hello Elastic Beanstalk!" Python application that you'll deploy using Elastic Beanstalk.

Create a text file named `application.py` in the directory you just created with the following contents.

Example ~/eb-python/application.py

```
from flask import Flask
application = Flask(__name__)

@application.route('/')
def hello_elastic_beanstalk():
        return 'Hello Elastic Beanstalk!'
```

Create a text file named `requirements.txt` with the following line. This file contains the required `pip` packages for the application to run.

Example ~/eb-python/requirements.txt

```
Flask
```

Step 2: Run your application locally

Run the following command to run your application locally.

```
~/eb-python$ export FLASK_APP=application.py && flask run --port 5000
```

You should see output similar to the following

```
Serving Flask app 'application.py'
Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a
 production WSGI server instead.
Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [01/Jan/1970 00:00:00] "GET / HTTP/1.1" 200 -
```

Navigate to `http://localhost:5000` in your web browser. The web browser should display “Hello Elastic Beanstalk!”.

Step 3: Deploy your Python application with the EB CLI

Run the following commands to create an Elastic Beanstalk environment for this application.

To create an environment and deploy your Python application

1. Initialize your EB CLI repository with the **eb init** command.

   ```
   ~/eb-python$ eb init -p python-3.9 python-tutorial --region us-east-2
   ```

   This command creates an application named `python-tutorial` and configures your local repository to create environments with the provided Python platform version.

2. (Optional) Run **eb init** again to configure a default key pair so that you can use SSH to connect to the EC2 instance running your application.

   ```
   ~/eb-python$ eb init
   Do you want to set up SSH for your instances?
   (y/n): y
   Select a keypair.
   1) my-keypair
   2) [ Create new KeyPair ]
   ```

   Select a key pair if you have one already, or follow the prompts to create one. If you don't see the prompt or need to change your settings later, run **eb init -i**.

3. Create an environment and deploy your application to it with **eb create**. Elastic Beanstalk automatically builds a zip file for your application and starts it on port 5000.

   ```
   ~/eb-python$ eb create python-env
   ```

   It takes about five minutes for Elastic Beanstalk to create your environment.

Step 4: Run your application on Elastic Beanstalk

When the process to create your environment completes, open your website with **eb open**.

```
~/eb-python$ eb open
```

Congratulations! You've deployed a Python application with Elastic Beanstalk! This opens a browser window using the domain name created for your application.

Step 5: Clean up

You can terminate your environment when you finish working with your application. Elastic Beanstalk terminates all AWS resources associated with your environment.

To terminate your Elastic Beanstalk environment with the EB CLI run the following command.

```
~/eb-python$ eb terminate
```

AWS resources for your application

You just created a single instance application. It serves as a straightforward sample application with a single EC2 instance, so it doesn't require load balancing or auto scaling. For single instance applications Elastic Beanstalk creates the following AWS resources:

- **EC2 instance** – An Amazon EC2 virtual machine configured to run web apps on the platform you choose.

  Each platform runs a different set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination thereof. Most platforms use either Apache or nginx as a reverse proxy that processes web traffic in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group** – An Amazon EC2 security group configured to allow incoming traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic is not allowed on other ports.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.

- **Domain name** – A domain name that routes to your web app in the form *subdomain.region.elasticbeanstalk.com*.

Elastic Beanstalk manages all of these resources. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Next steps

After you have an environment running an application, you can deploy a new version of the application or a different application at any time. Deploying a new application version is very quick because it doesn't require provisioning or restarting EC2 instances. You can also explore your new environment using the Elastic Beanstalk console. For detailed steps, see Explore your environment in the *Getting started* chapter of this guide.

Try more tutorials

If you'd like to try other tutorials with different example applications, see the following tutorials:

- Deploying a Flask application to Elastic Beanstalk
- Deploying a Django application to Elastic Beanstalk

After you deploy a sample application or two and are ready to start developing and running Python applications locally, see Setting up your Python development environment for Elastic Beanstalk.

Deploy with the Elastic Beanstalk console

You can also use the Elastic Beanstalk console to launch the sample application. For detailed steps, see Create an example application in the *Getting started* chapter of this guide.

Setting up your Python development environment for Elastic Beanstalk

This topic provides instructions to set up a Python development environment to test your application locally prior to deploying it to AWS Elastic Beanstalk. It also references websites that provide installation instructions for useful tools.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

Sections

Prerequisites

The following list provides the common prerequisites for working with Elastic Beanstalk and your Python applications:

- **Python language** – Install the version of the Python language that's included on your chosen Elastic Beanstalk Python platform version. For a list of our supported Python language versions, see Supported Python platforms in the *AWS Elastic Beanstalk Platforms* guide. If you don't already have Python set up on your development machine, see the Python downloads page on the Python website.
- **`pip` utility** – The `pip` utility is Python's package installer. It installs and lists dependencies for your project, so that Elastic Beanstalk knows how to set up your application's environment. For more information about `pip`, see the pip page on the *pip.pypa.io* website.
- **(Optional) The Elastic Beanstalk Command Line Interface (EB CLI)** – The EB CLI can package your application with the necessary deployment files. It can also create an Elastic Beanstalk environment and deploy your application to it. You can also make deployments via the Elastic Beanstalk console, so the EB CLI is not strictly necessary.
- **A working SSH installation** – You can connect to your running instances with the SSH protocol to examine or debug a deployment.
- **`virtualenv` package** – This `virtualenv` tool creates a development and test environment for your application. Elastic Beanstalk can replicate this environment without installing extra

packages that aren't required by your application. For more information, see the virtualenv website. After installing Python, you can install `virtualenv` package with the following command:

```
$ pip install virtualenv
```

Using a virtual environment

Once you have the prerequisites installed, set up a virtual environment with `virtualenv` to install your application's dependencies. By using a virtual environment, you can discern exactly which packages are needed by your application so that the required packages are installed on the EC2 instances that are running your application.

To set up a virtual environment

1. Open a command-line window and type:

   ```
   $ virtualenv /tmp/eb_python_app
   ```

   Replace *eb_python_app* with a name that makes sense for your application (using your application's name is a good idea). The `virtualenv` command creates a virtual environment for you in the specified directory and prints the results of its actions:

   ```
   Running virtualenv with interpreter /usr/bin/python
   New python executable in /tmp/eb_python_app/bin/python3.12
   Also creating executable in /tmp/eb_python_app/bin/python
   Installing setuptools, pip...done.
   ```

2. Once your virtual environment is ready, start it by running the `activate` script located in the environment's `bin` directory. For example, to start the **eb_python_app** environment created in the previous step, you would type:

   ```
   $ source /tmp/eb_python_app/bin/activate
   ```

   The virtual environment prints its name (for example: `(eb_python_app)`) at the beginning of each command prompt, reminding you that you're in a virtual Python environment.

3. To stop using your virtual environment and go back to the system's default Python interpreter with all its installed libraries, run the `deactivate` command.

   ```
   (eb_python_app) $ deactivate
   ```

> **Note**
>
> Once created, you can restart the virtual environment at any time by running its `activate` script again.

Configuring a Python project for Elastic Beanstalk

You can use the Elastic Beanstalk CLI to prepare your Python applications for deployment with Elastic Beanstalk.

To configure a Python application for deployment with Elastic Beanstalk

1. From within your virtual environment, return to the top of your project's directory tree (`python_eb_app`), and type:

   ```
   pip freeze >requirements.txt
   ```

   This command copies the names and versions of the packages that are installed in your virtual environment to `requirements.txt`, For example, if the *PyYAML* package, version *3.11* is installed in your virtual environment, the file will contain the line:

   ```
   PyYAML==3.11
   ```

   This allows Elastic Beanstalk to replicate your application's Python environment using the same packages and same versions that you used to develop and test your application.

2. Configure the EB CLI repository with the **eb init** command. Follow the prompts to choose a region, platform and other options.

By default, Elastic Beanstalk looks for a file called `application.py` to start your application. If this doesn't exist in the Python project that you've created, some adjustment of your application's environment is necessary. You will also need to set environment variables so that your application's modules can be loaded. See Using the Elastic Beanstalk Python platform for more information.

Using the Elastic Beanstalk Python platform

This topic describes how to configure, build, and run your Python applications on Elastic Beanstalk.

AWS Elastic Beanstalk supports a number of platform branches for different versions of the Python programming language. See Python in the *AWS Elastic Beanstalk Platforms* document for a full list.

The Python web applications can run behind a proxy server with WSGI. Elastic Beanstalk provides Gunicorn as the default WSGI server.

You can add a `Procfile` to your source bundle to specify and configure the WSGI server for your application. For details, see the section called “Procfile”.

You can use the `Pipfile` and `Pipfile.lock` files created by Pipenv to specify Python package dependencies and other requirements. For details about specifying dependencies, see the section called “Specifying dependencies”.

Elastic Beanstalk provides configuration options that you can use to customize the software that runs on the EC2 instances in your Elastic Beanstalk environment. You can configure environment variables needed by your application, enable log rotation to Amazon S3, and map folders in your application source that contain static files to paths served by the proxy server.

Configuration options are available in the Elastic Beanstalk console for modifying the configuration of a running environment. To avoid losing your environment's configuration when you terminate it, you can use saved configurations to save your settings and later apply them to another environment.

To save settings in your source code, you can include configuration files. Settings in configuration files are applied every time you create an environment or deploy your application. You can also use configuration files to install packages, run scripts, and perform other instance customization operations during deployments.

Settings applied in the Elastic Beanstalk console override the same settings in configuration files, if they exist. This lets you have default settings in configuration files, and override them with environment-specific settings in the console. For more information about precedence, and other methods of changing settings, see Configuration options.

For Python packages available from `pip`, you can include a requirements file in the root of your application source code. Elastic Beanstalk installs any dependency packages specified in a requirements file during deployment. For details, see the section called "Specifying dependencies".

For details about the various ways you can extend an Elastic Beanstalk Linux-based platform, see the section called "Extending Linux platforms".

Configuring your Python environment

The Python platform settings let you fine-tune the behavior of your Amazon EC2 instances. You can edit the Elastic Beanstalk environment's Amazon EC2 instance configuration using the Elastic Beanstalk console.

Use the Elastic Beanstalk console to configure Python process settings, enable AWS X-Ray, enable log rotation to Amazon S3, and configure variables that your application can read from the environment.

To configure your Python environment in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.

Python settings

- **Proxy server** – The proxy server to use on your environment instances. By default, nginx is used.
- **WSGI Path** – The name of or path to your main application file. For example, `application.py`, or `django/wsgi.py`.
- **NumProcesses** – The number of processes to run on each application instance.
- **NumThreads** – The number of threads to run in each process.

AWS X-Ray settings

- **X-Ray daemon** – Run the AWS X-Ray daemon to process trace data from the AWS X-Ray SDK for Python.

Log options

The Log Options section has two settings:

- **Instance profile**– Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3** – Specifies whether log files for your application's Amazon EC2 instances are copied to the Amazon S3 bucket associated with your application.

Static files

To improve performance, you can use the **Static files** section to configure the proxy server to serve static files (for example, HTML or images) from a set of directories inside your web application. For each directory, you set the virtual path to directory mapping. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application.

For details about configuring static files using configuration files or the Elastic Beanstalk console, see the section called "Static files".

By default, the proxy server in a Python environment serves any files in a folder named `static` at the `/static` path. For example, if your application source contains a file named `logo.png` in a folder named `static`, the proxy server serves it to users at *`subdomain`*`.elasticbeanstalk.com/static/logo.png`. You can configure additional mappings as explained in this section.

Environment properties

You can use environment properties to provide information to your application and configure environment variables. For example, you can create an environment property named `CONNECTION_STRING` that specifies a connection string that your application can use to connect to a database.

Inside the Python environment running in Elastic Beanstalk, these values are accessible using Python's `os.environ` dictionary. For more information, see http://docs.python.org/library/os.html.

You can use code that looks similar to the following to access the keys and parameters:

```
import os
endpoint = os.environ['API_ENDPOINT']
```

Environment properties can also provide information to a framework. For example, you can create a property named DJANGO_SETTINGS_MODULE to configure Django to use a specific settings module. Depending on the environment, the value could be development.settings, production.settings, etc.

See Environment variables and other software settings for more information.

Python configuration namespaces

You can use a configuration file to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be platform specific or apply to all platforms in the Elastic Beanstalk service as a whole. Configuration options are organized into *namespaces*.

The Python platform defines options in the aws:elasticbeanstalk:environment:proxy, aws:elasticbeanstalk:environment:proxy:staticfiles, and aws:elasticbeanstalk:container:python namespaces.

The following example configuration file specifies configuration option settings to create an environment property named DJANGO_SETTINGS_MODULE, choose the Apache proxy server, specify two static files options that map a directory named statichtml to the path /html and a directory named staticimages to the path /images, and specify additional settings in the aws:elasticbeanstalk:container:python namespace. This namespace contains options that let you specify the location of the WSGI script in your source code, and the number of threads and processes to run in WSGI.

```
option_settings:
  aws:elasticbeanstalk:application:environment:
    DJANGO_SETTINGS_MODULE: production.settings
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: apache
  aws:elasticbeanstalk:environment:proxy:staticfiles:
    /html: statichtml
    /images: staticimages
  aws:elasticbeanstalk:container:python:
    WSGIPath: ebdjango.wsgi:application
    NumProcesses: 3
    NumThreads: 20
```

Notes

- If you're using an Amazon Linux AMI Python platform version (preceding Amazon Linux 2), replace the value for `WSGIPath` with `ebdjango/wsgi.py`. The value in the example works with the Gunicorn WSGI server, which isn't supported on Amazon Linux AMI platform versions.
- In addition, these older platform versions use a different namespace for configuring static files—`aws:elasticbeanstalk:container:python:staticfiles`. It has the same option names and semantics as the standard static file namespace.

Configuration files also support several keys to further modify the software on your environment's instances. This example uses the packages key to install Memcached with `yum` and container commands to run commands that configure the server during deployment:

```
packages:
  yum:
    libmemcached-devel: '0.31'

container_commands:
  collectstatic:
    command: "django-admin.py collectstatic --noinput"
  01syncdb:
    command: "django-admin.py syncdb --noinput"
    leader_only: true
  02migrate:
    command: "django-admin.py migrate"
    leader_only: true
  03wsgipass:
    command: 'echo "WSGIPassAuthorization On" >> ../wsgi.conf'
  99customize:
    command: "scripts/customize.sh"
```

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See Configuration options for more information.

The python3 executable

The version of the python3 executable available on EC2 instances in Elastic Beanstalk Python environments will not always correspond to the same Python version used by the platform. For example, on the Python 3.12 AL2023 platform, /usr/bin/python3 points to Python 3.9. This is because Python 3.9 is the *system Python* on AL2023. For more information, see Python in AL2023 in the *Amazon Linux 2023 User Guide*. You can access an executable corresponding to the Python version used by the platform at a versioned location (e.g. /usr/bin/python3.12) or in the application virtual environment bin directory (e.g. /var/app/venv/staging-LQM1lest/bin/python3). The platform uses the correct Python executable that corresponds to the platform branch.

Configuring the WSGI server with a Procfile on Elastic Beanstalk

You can add a Procfile to your source bundle to specify and configure the WSGI server for your application. You can specify custom start and run commands in the Procfile.

When you use a Procfile, it overrides aws:elasticbeanstalk:container:python namespace options that you set using configuration files.

The following example uses a Procfile to specify uWSGI as the server and configure it.

Example Procfile

```
web: uwsgi --http :8000 --wsgi-file application.py --master --processes 4 --threads 2
```

The following example uses a Procfile to configure Gunicorn, the default WSGI server.

Example Procfile

```
web: gunicorn --bind :8000 --workers 3 --threads 2 project.wsgi:application
```

Notes

- If you configure any WSGI server other than Gunicorn, be sure to also specify it as a dependency of your application, so that it is installed on your environment instances. For details about dependency specification, see the section called "Specifying dependencies".

- The default port for the WSGI server is 8000. If you specify a different port number in your `Procfile` command, set the `PORT` environment property to this port number too.

Specifying dependencies using a requirements file on Elastic Beanstalk

This topic describes how to configure you application to install other Python packages that it requires. A typical Python application has dependencies on other third-party Python packages. With the Elastic Beanstalk Python platform, you have multiple ways to specify Python packages that your application depends on.

Use pip and `requirements.txt`

The standard tool for installing Python packages is `pip`. It has a feature that allows you to specify all the packages you need (as well as their versions) in a single requirements file. For more information about the requirements file, see Requirements File Format on the pip documentation website.

Create a file named `requirements.txt` and place it in the top-level directory of your source bundle. The following is an example `requirements.txt` file for Django.

```
Django==2.2
mysqlclient==2.0.3
```

In your development environment, you can use the `pip freeze` command to generate your requirements file.

```
~/my-app$ pip freeze > requirements.txt
```

To ensure that your requirements file only contains packages that are actually used by your application, use a virtual environment that only has those packages installed. Outside of a virtual environment, the output of `pip freeze` will include all `pip` packages installed on your development machine, including those that came with your operating system.

Note

On Amazon Linux AMI Python platform versions, Elastic Beanstalk doesn't natively support Pipenv or Pipfiles. If you use Pipenv to manage your application's dependencies, run the following command to generate a `requirements.txt` file.

```
~/my-app$ pipenv lock -r > requirements.txt
```

To learn more, see Generating a requirements.txt in the Pipenv documentation.

Use Pipenv and Pipfile

Pipenv is a modern Python packaging tool. It combines package installation with the creation and management of a dependency file and a virtualenv for your application. For more information, see Pipenv: Python Dev Workflow for Humans.

Pipenv maintains two files:

- `Pipfile` — This file contains various types of dependencies and requirements.
- `Pipfile.lock` — This file contains a version snapshot that enables deterministic builds.

You can create these files on your development environment and include them in the top-level directory of the source bundle that you deploy to Elastic Beanstalk. For more information about these two files, see Example Pipfile and Pipfile.lock.

The following example uses Pipenv to install Django and the Django REST framework. These commands create the files `Pipfile` and `Pipfile.lock`.

```
~/my-app$ pipenv install django
~/my-app$ pipenv install djangorestframework
```

Precedence

If you include more than one of the requirements files described in this topic, Elastic Beanstalk uses just one of them. The following list shows the precedence, in descending order.

1. `requirements.txt`
2. `Pipfile.lock`
3. `Pipfile`

Note

Starting with the March 7, 2023 Amazon Linux 2 platform release, if you provide more than one of these files, Elastic Beanstalk will issue a console message stating which one of the dependency files was used during a deployment.

The following steps describe the logic that Elastic Beanstalk follows to install the dependencies when it's deploying an instance.

- If there is a `requirements.txt` file, we use the command `pip install -r requirements.txt`.
- Starting with the March 7, 2023 Amazon Linux 2 platform release, if there is no `requirements.txt` file, but there is a `Pipfile.lock`, we use the command `pipenv sync`. Prior to that release, we used `pipenv install --ignore-pipfile`.
- If there is neither a `requirements.txt` file nor a `Pipfile.lock`, but there is a `Pipfile`, we use the command `pipenv install --skip-lock`.
- If none of the three requirements files are found, we don't install any application dependencies.

Deploying a Flask application to Elastic Beanstalk

This tutorial walks you through the process of generating a Flask application and deploying it to an AWS Elastic Beanstalk environment. Flask is an open source web application framework for Python.

In this tutorial, you'll do the following:

- Set up a Python virtual environment with Flask
- Create a Flask application
- Deploy your site with the EB CLI
- Cleanup

Prerequisites

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in Learn how to get started with Elastic Beanstalk to launch your first Elastic Beanstalk environment.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

In this tutorial we use Python 3.11 and the corresponding Elastic Beanstalk platform version. Install Python by following the instructions at Setting up your Python development environment for Elastic Beanstalk.

The Flask framework will be installed as part of the tutorial.

This tutorial also uses the Elastic Beanstalk Command Line Interface (EB CLI). For details on installing and configuring the EB CLI, see Install EB CLI with setup script (recommended) and Configure the EB CLI.

Set up a Python virtual environment with Flask

Create a project directory and virtual environment for your application, and install Flask.

To set up your project environment

1. Create a project directory.

   ```
   ~$ mkdir eb-flask
   ~$ cd eb-flask
   ```

2. Create and activate a virtual environment named `virt`:

   ```
   ~/eb-flask$ virtualenv virt
   ~$ source virt/bin/activate
   (virt) ~/eb-flask$
   ```

   You will see `(virt)` prepended to your command prompt, indicating that you're in a virtual environment. Use the virtual environment for the rest of this tutorial.

3. Install Flask with `pip install`:

```
(virt)~/eb-flask$ pip install flask==3.0.3
```

4. View the installed libraries with `pip freeze`:

   ```
   (virt)~/eb-flask$ pip freeze
   blinker==1.8.2
   click==8.1.7
   Flask==3.0.3
   importlib_metadata==8.5.0
   itsdangerous==2.2.0
   Jinja2==3.1.4
   MarkupSafe==2.1.5
   Werkzeug==3.0.4
   zipp==3.20.2
   ```

   This command lists all of the packages installed in your virtual environment. Because you are in a virtual environment, globally installed packages like the EB CLI are not shown.

5. Save the output from `pip freeze` to a file named `requirements.txt`.

   ```
   (virt)~/eb-flask$ pip freeze > requirements.txt
   ```

   This file tells Elastic Beanstalk to install the libraries during deployment. For more information, see Specifying dependencies using a requirements file on Elastic Beanstalk.

Create a Flask application

Next, create an application that you'll deploy using Elastic Beanstalk. We'll create a "Hello World" RESTful web service.

Create a new text file in this directory named `application.py` with the following contents:

Example ~/eb-flask/application.py

```
from flask import Flask

# print a nice greeting.
def say_hello(username = "World"):
    return '<p>Hello %s!</p>\n' % username
```

```
# some bits of text for the page.
header_text = '''
    <html>\n<head> <title>EB Flask Test</title> </head>\n<body>'''
instructions = '''
    <p><em>Hint</em>: This is a RESTful web service! Append a username
    to the URL (for example: <code>/Thelonious</code>) to say hello to
    someone specific.</p>\n'''
home_link = '<p><a href="/">Back</a></p>\n'
footer_text = '</body>\n</html>'

# EB looks for an 'application' callable by default.
application = Flask(__name__)

# add a rule for the index page.
application.add_url_rule('/', 'index', (lambda: header_text +
    say_hello() + instructions + footer_text))

# add a rule when the page is accessed with a name appended to the site
# URL.
application.add_url_rule('/<username>', 'hello', (lambda username:
    header_text + say_hello(username) + home_link + footer_text))

# run the app.
if __name__ == "__main__":
    # Setting debug to True enables debug output. This line should be
    # removed before deploying a production app.
    application.debug = True
    application.run()
```

This example prints a customized greeting that varies based on the path used to access the service.

Note

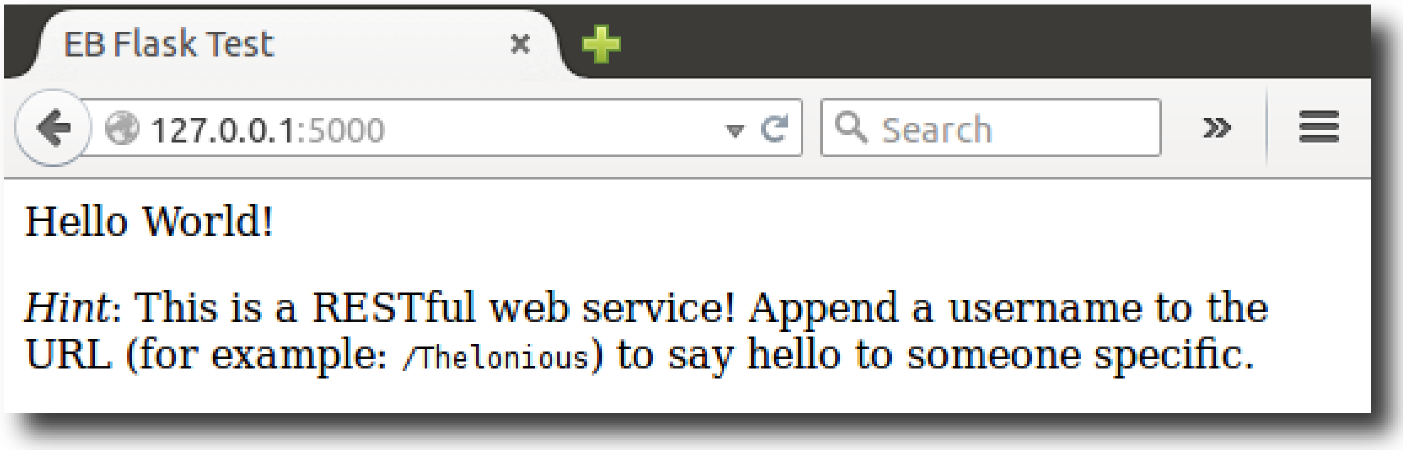
By adding `application.debug = True` before running the application, debug output is enabled in case something goes wrong. It's a good practice for development, but you should remove debug statements in production code, since debug output can reveal internal aspects of your application.

Using `application.py` as the filename and providing a callable `application` object (the Flask object, in this case) allows Elastic Beanstalk to easily find your application's code.

Run `application.py` with Python:

```
(virt) ~/eb-flask$ python application.py
 * Serving Flask app "application" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 313-155-123
```

Open `http://127.0.0.1:5000/` in your web browser. You should see the application running, showing the index page:



Hello World!

Hint: This is a RESTful web service! Append a username to the URL (for example: `/Thelonious`) to say hello to someone specific.

Check the server log to see the output from your request. You can stop the web server and return to your virtual environment by typing **Ctrl+C**.

If you got debug output instead, fix the errors and make sure the application is running locally before configuring it for Elastic Beanstalk.

Deploy your site with the EB CLI

You've added everything you need to deploy your application on Elastic Beanstalk. Your project directory should now look like this:

```
~/eb-flask/
|-- virt
|-- application.py
```

```
`-- requirements.txt
```

The `virt` folder, however, is not required for the application to run on Elastic Beanstalk. When you deploy, Elastic Beanstalk creates a new virtual environment on the server instances and installs the libraries listed in `requirements.txt`. To minimize the size of the source bundle that you upload during deployment, add an .ebignore file that tells the EB CLI to leave out the `virt` folder.

Example ~/eb-flask/.ebignore

```
virt
```

Next, you'll create your application environment and deploy your configured application with Elastic Beanstalk.

To create an environment and deploy your Flask application

1. Initialize your EB CLI repository with the **eb init** command:

   ```
   ~/eb-flask$ eb init -p python-3.11 flask-tutorial --region us-east-2
   Application flask-tutorial has been created.
   ```

   This command creates a new application named `flask-tutorial` and configures your local repository to create environments with the latest Python 3.11 platform version.

2. (optional) Run **eb init** again to configure a default keypair so that you can connect to the EC2 instance running your application with SSH:

   ```
   ~/eb-flask$ eb init
   Do you want to set up SSH for your instances?
   (y/n): y
   Select a keypair.
   1) my-keypair
   2) [ Create new KeyPair ]
   ```

   Select a key pair if you have one already, or follow the prompts to create a new one. If you don't see the prompt or need to change your settings later, run **eb init -i**.

3. Create an environment and deploy your application to it with **eb create**:

   ```
   ~/eb-flask$ eb create flask-env
   ```

Environment creation takes about 5 minutes and creates the following resources:

- **EC2 instance –** An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

  Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or NGINX as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group –** An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer –** An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group –** An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group –** An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket –** A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms –** Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack –** Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name –** A domain name that routes to your web app in the form *`subdomain`.`region`.elasticbeanstalk.com*.

  Domain security

  To augment the security of your Elastic Beanstalk applications, the *elasticbeanstalk.com* domain is registered in the Public Suffix List (PSL).

If you ever need to set sensitive cookies in the default domain name for your Elastic Beanstalk applications, we recommend that you use cookies with a `__Host-` prefix for increased security. This practice defends your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

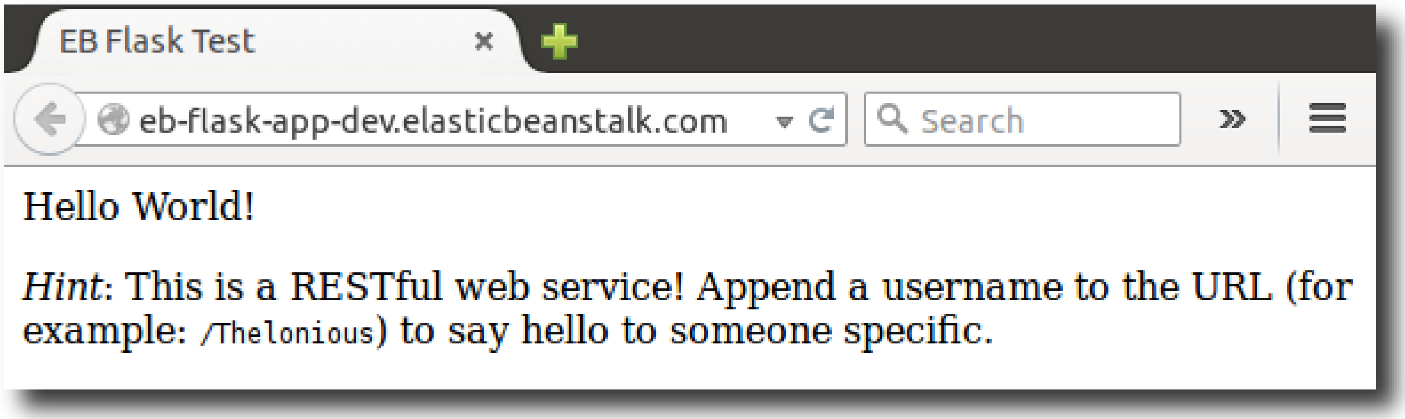
Note

The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see Using Elastic Beanstalk with Amazon S3.

When the environment creation process completes, open your web site with **eb open**:

```
~/eb-flask$ eb open
```

This will open a browser window using the domain name created for your application. You should see the same Flask website that you created and tested locally.



Hello World!

Hint: This is a RESTful web service! Append a username to the URL (for example: `/Thelonious`) to say hello to someone specific.

If you don't see your application running, or get an error message, see Troubleshooting Deployments for help with how to determine the cause of the error.

If you *do* see your application running, then congratulations, you've deployed your first Flask application with Elastic Beanstalk!

Cleanup

After you finish working with the demo code, you can terminate your environment. Elastic Beanstalk deletes all related AWS resources, such as Amazon EC2 instances, database instances, load balancers, security groups, and alarms.

Removing resources does not delete the Elastic Beanstalk application, so you can create new environments for your application at any time.

To terminate your Elastic Beanstalk environment from the console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

Or, with the EB CLI:

```
~/eb-flask$ eb terminate flask-env
```

Next steps

For more information about Flask, visit flask.pocoo.org.

If you'd like to try out another Python web framework, check out Deploying a Django application to Elastic Beanstalk.

Deploying a Django application to Elastic Beanstalk

This tutorial walks through the deployment of a default, autogenerated Django website to an AWS Elastic Beanstalk environment running Python. This tutorial shows you how to host a Python web app in the cloud by using an Elastic Beanstalk environment.

In this tutorial, you'll do the following:

- Set up a Python virtual environment and install Django
- Create a Django project
- Configure your Django application for Elastic Beanstalk
- Deploy your site with the EB CLI
- Update your application
- Clean up

Prerequisites

To follow this tutorial, you should have all of the Common Prerequisites for Python installed, including the following packages:

- Python 3.7 or later
- `pip`
- `virtualenv`
- `awsebcli`

The Django framework is installed as part of the tutorial.

Note

Creating environments with the EB CLI requires a service role. You can create a service role by creating an environment in the Elastic Beanstalk console. If you don't have a service role, the EB CLI attempts to create one when you run `eb create`.

Set up a Python virtual environment and install Django

Create a virtual environment with `virtualenv` and use it to install Django and its dependencies. By using a virtual environment, you can know exactly which packages your application needs, so that the required packages are installed on the Amazon EC2 instances that are running your application.

The following steps demonstrate the commands you must enter for Unix-based systems and Windows, shown on separate tabs.

To set up your virtual environment

1. Create a virtual environment named `eb-virt`.

   Unix-based systems

   ```
   ~$ virtualenv ~/eb-virt
   ```

   Windows

   ```
   C:\> virtualenv %HOMEPATH%\eb-virt
   ```

2. Activate the virtual environment.

   Unix-based systems

   ```
   ~$ source ~/eb-virt/bin/activate
   (eb-virt) ~$
   ```

   Windows

   ```
   C:\>%HOMEPATH%\eb-virt\Scripts\activate
   (eb-virt) C:\>
   ```

   You'll see `(eb-virt)` prepended to your command prompt, indicating that you're in a virtual environment.

   Note

   The rest of these instructions show the Linux command prompt in your home directory ~$. On Windows this is `C:\Users\`*`USERNAME`*>, where *USERNAME* is your Windows login name.

3. Use `pip` to install Django.

   ```
   (eb-virt)~$ pip install django==2.2
   ```

Note

The Django version you install must be compatible with the Python version on the Elastic Beanstalk Python configuration that you choose for deploying your application. For information about deployment, see ??? in this topic.
For more information about current Python platform versions, see Python in the *AWS Elastic Beanstalk Platforms* document.
For Django version compatibility with Python, see What Python version can I use with Django?

4. To verify that Django is installed, enter the following.

```
(eb-virt)~$ pip freeze
Django==2.2
...
```

This command lists all of the packages installed in your virtual environment. Later, you use the output of this command to configure your project for use with Elastic Beanstalk.

Create a Django project

Now you are ready to create a Django project and run it on your machine, using the virtual environment.

Note

This tutorial uses SQLite, which is a database engine included in Python. The database is deployed with your project files. For production environments, we recommend that you use Amazon Relational Database Service (Amazon RDS), and that you separate it from your environment. For more information, see Adding an Amazon RDS DB instance to your Python Elastic Beanstalk environment.

To generate a Django application

1. Activate your virtual environment.

Unix-based systems

```
~$ source ~/eb-virt/bin/activate
(eb-virt) ~$
```

Windows

```
C:\>%HOMEPATH%\eb-virt\Scripts\activate
(eb-virt) C:\>
```

You'll see the `(eb-virt)` prefix prepended to your command prompt, indicating that you're in a virtual environment.

Note

The rest of these instructions show the Linux command prompt ~$ in your home directory and the Linux home directory ~/. On Windows these are `C:\Users\`*`USERNAME`*>, where *USERNAME* is your Windows login name.

2. Use the `django-admin startproject` command to create a Django project named `ebdjango`.

```
(eb-virt)~$ django-admin startproject ebdjango
```

This command creates a standard Django site named **ebdjango** with the following directory structure.

```
~/ebdjango
  |-- ebdjango
  |   |-- __init__.py
  |   |-- settings.py
  |   |-- urls.py
  |   `-- wsgi.py
  `-- manage.py
```

3. Run your Django site locally with `manage.py runserver`.

```
(eb-virt) ~$ cd ebdjango
```

```
(eb-virt) ~/ebdjango$ python manage.py runserver
```

4. In a web browser, open `http://127.0.0.1:8000/` to view the site.
5. Check the server log to see the output from your request. To stop the web server and return to your virtual environment, press **Ctrl+C**.

```
Django version 2.2, using settings 'ebdjango.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
[07/Sep/2018 20:14:09] "GET / HTTP/1.1" 200 16348
Ctrl+C
```

Configure your Django application for Elastic Beanstalk

Now that you have a Django-powered site on your local machine, you can configure it for deployment with Elastic Beanstalk.

By default, Elastic Beanstalk looks for a file named `application.py` to start your application. Because this doesn't exist in the Django project that you've created, you need to make some adjustments to your application's environment. You also must set environment variables so that your application's modules can be loaded.

To configure your site for Elastic Beanstalk

1. Activate your virtual environment.

   Unix-based systems

   ```
   ~/ebdjango$ source ~/eb-virt/bin/activate
   ```

   Windows

   ```
   C:\Users\USERNAME\ebdjango>%HOMEPATH%\eb-virt\Scripts\activate
   ```

2. Run `pip freeze`, and then save the output to a file named `requirements.txt`.

```
(eb-virt) ~/ebdjango$ pip freeze > requirements.txt
```

Elastic Beanstalk uses `requirements.txt` to determine which package to install on the EC2 instances that run your application.

3. Create a directory named `.ebextensions`.

```
(eb-virt) ~/ebdjango$ mkdir .ebextensions
```

4. In the `.ebextensions` directory, add a configuration file named `django.config` with the following text.

   Example ~/ebdjango/.ebextensions/django.config

```
option_settings:
  aws:elasticbeanstalk:container:python:
    WSGIPath: ebdjango.wsgi:application
```

   This setting, `WSGIPath`, specifies the location of the WSGI script that Elastic Beanstalk uses to start your application.

   Note

   If you're using an Amazon Linux AMI Python platform version (preceding Amazon Linux 2), replace the value for `WSGIPath` with `ebdjango/wsgi.py`. The value in the example works with the Gunicorn WSGI server, which isn't supported on Amazon Linux AMI platform versions.

5. Deactivate your virtual environment with the `deactivate` command.

```
(eb-virt) ~/ebdjango$ deactivate
```

   Reactivate your virtual environment whenever you need to add packages to your application or run your application locally.

Deploy your site with the EB CLI

You've added everything you need to deploy your application on Elastic Beanstalk. Your project directory should now look like this.

```
~/ebdjango/
|-- .ebextensions
|   `-- django.config
|-- ebdjango
|   |-- __init__.py
|   |-- settings.py
|   |-- urls.py
|   `-- wsgi.py
|-- db.sqlite3
|-- manage.py
`-- requirements.txt
```

Next, you'll create your application environment and deploy your configured application with Elastic Beanstalk.

Immediately after deployment, you'll edit Django's configuration to add the domain name that Elastic Beanstalk assigned to your application to Django's `ALLOWED_HOSTS`. Then you'll redeploy your application. This is a Django security requirement, designed to prevent HTTP `Host` header attacks. For more information, see Host header validation.

To create an environment and deploy your Django application

Note

This tutorial uses the EB CLI as a deployment mechanism, but you can also use the Elastic Beanstalk console to deploy a .zip file containing your project's contents.

1. Initialize your EB CLI repository with the **eb init** command.

   ```
   ~/ebdjango$ eb init -p python-3.7 django-tutorial
   Application django-tutorial has been created.
   ```

   This command creates an application named `django-tutorial`. It also configures your local repository to create environments with the latest Python 3.7 platform version.

2. (Optional) Run **eb init** again to configure a default key pair so that you can use SSH to connect to the EC2 instance running your application.

```
~/ebdjango$ eb init
Do you want to set up SSH for your instances?
(y/n): y
Select a keypair.
1) my-keypair
2) [ Create new KeyPair ]
```

   Select a key pair if you have one already, or follow the prompts to create one. If you don't see the prompt or need to change your settings later, run **eb init -i**.

3. Create an environment and deploy your application to it with **eb create**.

```
~/ebdjango$ eb create django-env
```

> **Note**
>
> If you see a "service role required" error message, run `eb create` interactively (without specifying an environment name) and the EB CLI creates the role for you.

   This command creates a load-balanced Elastic Beanstalk environment named `django-env`. Creating an environment takes about 5 minutes. As Elastic Beanstalk creates the resources needed to run your application, it outputs informational messages that the EB CLI relays to your terminal.

4. When the environment creation process completes, find the domain name of your new environment by running **eb status**.

```
~/ebdjango$ eb status
Environment details for: django-env
  Application name: django-tutorial
  ...
  CNAME: eb-django-app-dev.elasticbeanstalk.com
  ...
```

   Your environment's domain name is the value of the `CNAME` property.

5. Open the `settings.py` file in the `ebdjango` directory. Locate the `ALLOWED_HOSTS` setting, and then add your application's domain name that you found in the previous step to the setting's value. If you can't find this setting in the file, add it to a new line.

   ```
   ...
   ALLOWED_HOSTS = ['eb-django-app-dev.elasticbeanstalk.com']
   ```

6. Save the file, and then deploy your application by running **eb deploy**. When you run **eb deploy**, the EB CLI bundles up the contents of your project directory and deploys it to your environment.

   ```
   ~/ebdjango$ eb deploy
   ```

   Note

   If you are using Git with your project, see Using the EB CLI with Git.

7. When the environment update process completes, open your website with **eb open**.

   ```
   ~/ebdjango$ eb open
   ```

   This opens a browser window using the domain name created for your application. You should see the same Django website that you created and tested locally.

If you don't see your application running, or get an error message, see Troubleshooting deployments for help with how to determine the cause of the error.

If you *do* see your application running, then congratulations, you've deployed your first Django application with Elastic Beanstalk!

Update your application

Now that you have a running application on Elastic Beanstalk, you can update and redeploy your application or its configuration, and Elastic Beanstalk does the work of updating your instances and starting your new application version.

For this example, we'll enable Django's admin console and configure a few other settings.

Modify your site settings

By default, your Django website uses the UTC time zone to display time. You can change this by specifying a time zone in `settings.py`.

To change your site's time zone

1. Modify the `TIME_ZONE` setting in `settings.py`.

   Example ~/ebdjango/ebdjango/settings.py

   ```
   ...
   # Internationalization
   LANGUAGE_CODE = 'en-us'
   TIME_ZONE = 'US/Pacific'
   USE_I18N = True
   USE_L10N = True
   USE_TZ = True
   ```

   For a list of time zones, visit this page.
2. Deploy the application to your Elastic Beanstalk environment.

   ```
   ~/ebdjango/$ eb deploy
   ```

Create a site administrator

You can create a site administrator for your Django application to access the admin console directly from the website. Administrator login details are stored securely in the local database image included in the default project that Django generates.

To create a site administrator

1. Initialize your Django application's local database.

   ```
   (eb-virt) ~/ebdjango$ python manage.py migrate
   Operations to perform:
     Apply all migrations: admin, auth, contenttypes, sessions
   Running migrations:
     Applying contenttypes.0001_initial... OK
     Applying auth.0001_initial... OK
   ```

```
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying sessions.0001_initial... OK
```

2. Run `manage.py createsuperuser` to create an administrator.

```
(eb-virt) ~/ebdjango$ python manage.py createsuperuser
Username: admin
Email address: me@mydomain.com
Password: ********
Password (again): ********
Superuser created successfully.
```

3. To tell Django where to store static files, define `STATIC_ROOT` in `settings.py`.

Example ~/ebdjango/ebdjango/settings.py

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.2/howto/static-files/
STATIC_URL = '/static/'
STATIC_ROOT = 'static'
```

4. Run `manage.py collectstatic` to populate the `static` directory with static assets (JavaScript, CSS, and images) for the admin site.

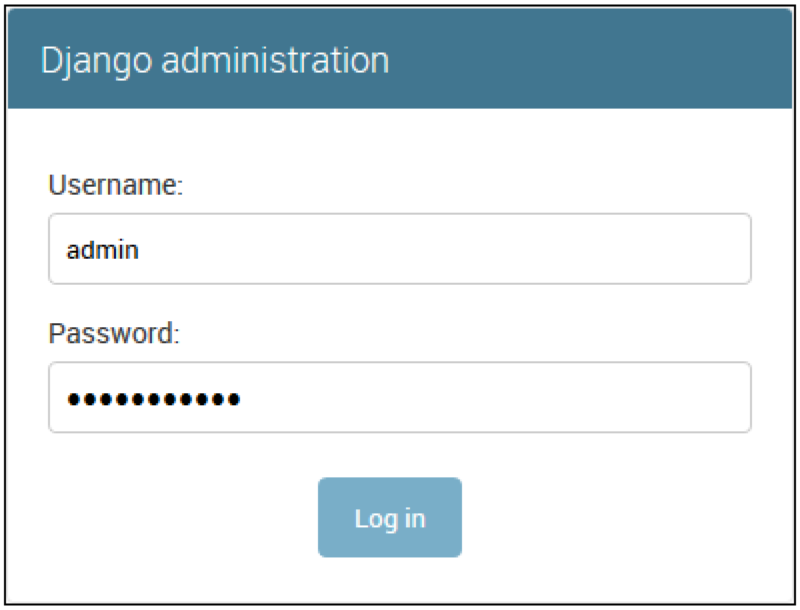
```
(eb-virt) ~/ebdjango$ python manage.py collectstatic
119 static files copied to ~/ebdjango/static
```

5. Deploy your application.

```
~/ebdjango$ eb deploy
```

6. View the admin console by opening the site in your browser, appending `/admin/` to the site URL, such as the following.

```
http://djang-env.p33kq46sfh.us-west-2.elasticbeanstalk.com/admin/
```


Django administration

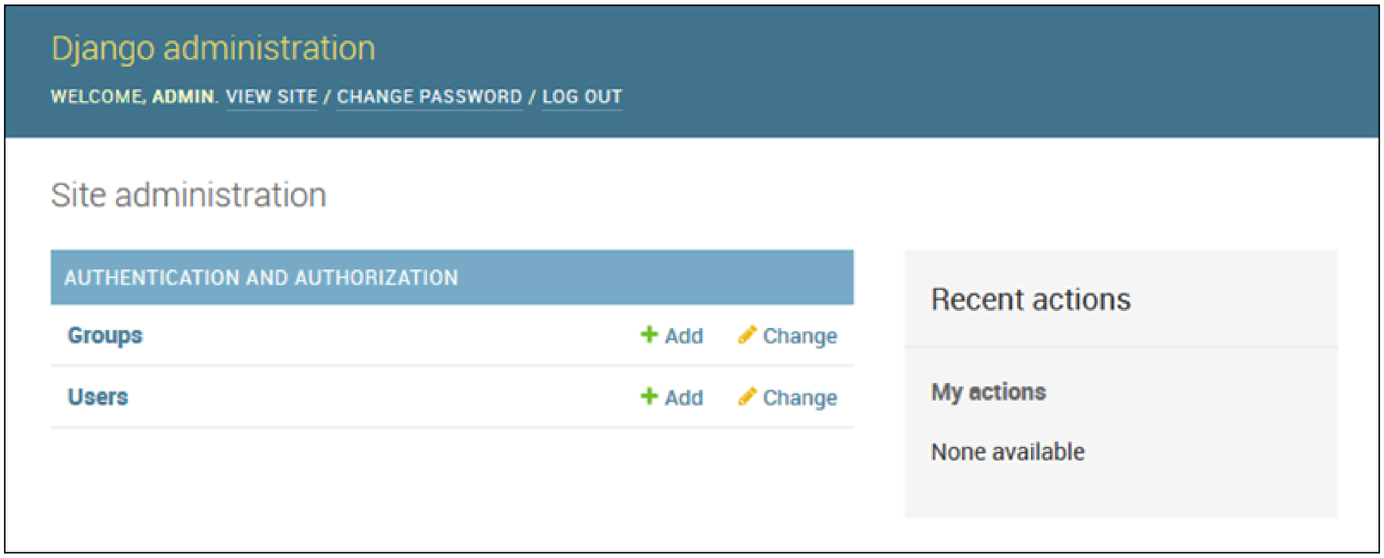
Username:

admin

Password:

●●●●●●●●●●●

7. Log in with the username and password that you configured in step 2.


Django administration

WELCOME, **ADMIN**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

Recent actions

My actions

None available

You can use a similar procedure of local updating/testing followed by **eb deploy**. Elastic Beanstalk does the work of updating your live servers, so you can focus on application development instead of server administration!

Add a database migration configuration file

You can add commands to your `.ebextensions` script that are run when your site is updated. This enables you to automatically generate database migrations.

To add a migrate step when your application is deployed

1. Create a configuration file named `db-migrate.config` with the following content.

   Example ~/ebdjango/.ebextensions/db-migrate.config

   ```
   container_commands:
     01_migrate:
       command: "source /var/app/venv/*/bin/activate && python3 manage.py migrate"
       leader_only: true
   option_settings:
     aws:elasticbeanstalk:application:environment:
       DJANGO_SETTINGS_MODULE: ebdjango.settings
   ```

   This configuration file activates the server's virtual environment and runs the `manage.py migrate` command during the deployment process, before starting your application. Because it runs before the application starts, you must also configure the `DJANGO_SETTINGS_MODULE` environment variable explicitly (usually `wsgi.py` takes care of this for you during startup). Specifying `leader_only: true` in the command ensures that it is run only once when you're deploying to multiple instances.

2. Deploy your application.

   ```
   ~/ebdjango$ eb deploy
   ```

Clean up

To save instance hours and other AWS resources between development sessions, terminate your Elastic Beanstalk environment with **eb terminate**.

```
~/ebdjango$ eb terminate django-env
```

This command terminates the environment and all of the AWS resources that run within it. It doesn't delete the application, however, so you can always create more environments with the same configuration by running **eb create** again.

If you're done with the sample application, you can also remove the project folder and virtual environment.

```
~$ rm -rf ~/eb-virt
~$ rm -rf ~/ebdjango
```

Next steps

For more information about Django, including an in-depth tutorial, see the official documentation.

If you want to try out another Python web framework, check out Deploying a Flask application to Elastic Beanstalk.

Adding an Amazon RDS DB instance to your Python Elastic Beanstalk environment

This topic provides instructions to create an Amazon RDS using the Elastic Beanstalk console. You can use an Amazon Relational Database Service (Amazon RDS) DB instance to store data gathered and modified by your application. The database can be coupled to your environment and managed by Elastic Beanstalk, or it can be created as decoupled and managed externally by another service. In these instructions the database is coupled to your environment and managed by Elastic Beanstalk. For more information about integrating an Amazon RDS with Elastic Beanstalk, see Adding a database to your Elastic Beanstalk environment.

Sections

Adding a DB instance to your environment

To add a DB instance to your environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Choose a DB engine, and enter a user name and password.
6. To save the changes choose **Apply** at the bottom of the page.

Adding a DB instance takes about 10 minutes. When the environment update is complete, the DB instance's hostname and other connection information are available to your application through the following environment properties:

Property name	Description	Property value
`RDS_HOSTNAME`	The hostname of the DB instance.	On the **Connectivity & security** tab on the Amazon RDS console: **Endpoint**.
`RDS_PORT`	The port where the DB instance accepts connectio ns. The default value varies among DB engines.	On the **Connectivity & security** tab on the Amazon RDS console: **Port**.
`RDS_DB_NAME`	The database name, **ebdb**.	On the **Configuration** tab on the Amazon RDS console: **DB Name**.
`RDS_USERNAME`	The username that you configured for your database.	On the **Configuration** tab on the Amazon RDS console: **Master username**.
`RDS_PASSWORD`	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

For more information about configuring a database instance coupled with an Elastic Beanstalk environment, see Adding a database to your Elastic Beanstalk environment.

Downloading a driver

Add the database driver to your project's requirements file.

Example requirements.txt – Django with MySQL

```
Django==2.2
mysqlclient==2.0.3
```

Common driver packages for Python

- **MySQL** – `mysqlclient`
- **PostgreSQL** – `psycopg2`
- **Oracle** – `cx_Oracle`
- **SQL Server** – `adodbapi`

For more information see Python DatabaseInterfaces and Django 2.2 - supported databases.

Connecting to a database

Elastic Beanstalk provides connection information for attached DB instances in environment properties. Use `os.environ['VARIABLE']` to read the properties and configure a database connection.

Example Django settings file – DATABASES dictionary

```
import os

if 'RDS_HOSTNAME' in os.environ:
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.mysql',
            'NAME': os.environ['RDS_DB_NAME'],
            'USER': os.environ['RDS_USERNAME'],
            'PASSWORD': os.environ['RDS_PASSWORD'],
            'HOST': os.environ['RDS_HOSTNAME'],
            'PORT': os.environ['RDS_PORT'],
        }
    }
```

Python tools and resources

There are several places you can go to get additional help when developing your Python applications:

Resource	Description
AWS SDK for Python (Boto3) on GitHub	Install Boto3 sourced from GitHub.
AWS SDK for Python (Boto3) homepage	The AWS SDK for Python (Boto3) homepage.
Python Developer Center	One-stop shop for sample code, documentation, tools, and additional resources.

Deploying Ruby applications with Elastic Beanstalk

This chapter provides instructions for configuring and deploying your Ruby web application to AWS Elastic Beanstalk. Elastic Beanstalk makes it easy to deploy, manage, and scale your Ruby web applications using Amazon Web Services.

You can deploy your application in just a few minutes using the Elastic Beanstalk Command Line Interface (EB CLI) or by using the Elastic Beanstalk console. After you deploy your Elastic Beanstalk application, you can continue to use the EB CLI to manage your application and environment, or you can use the Elastic Beanstalk console, AWS CLI, or the APIs.

This chapter also provides step-by-step instructions for deploying a sample application to Elastic Beanstalk using the EB CLI, and then updating the application to use the Rails and Sinatra web application frameworks.

The topics in this chapter assume that you have some knowledge of Elastic Beanstalk environments. If you haven't used Elastic Beanstalk before, try the getting started tutorial to learn the basics.

Topics

Setting up your Ruby development environment for Elastic Beanstalk

This chapter provides instructions to set up a Ruby development environment to test your application locally prior to deploying it to AWS Elastic Beanstalk. It also references websites that provide installation instructions for useful tools.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

Sections

Installing Ruby

Install GCC if you don't have a C compiler. On Ubuntu, use `apt`.

```
~$ sudo apt install gcc
```

On Amazon Linux, use `yum`.

```
~$ sudo yum install gcc
```

Install RVM to manage Ruby language installations on your machine. Use the commands at rvm.io to get the project keys and run the installation script.

```
~$ gpg2 --recv-keys key1 key2
~$ curl -sSL https://get.rvm.io | bash -s stable
```

This script installs RVM in a folder named `.rvm` in your user directory, and modifies your shell profile to load a setup script whenever you open a new terminal. Load the script manually to get started.

```
~$ source ~/.rvm/scripts/rvm
```

Use `rvm get head` to get the latest version.

```
~$ rvm get head
```

View the available versions of Ruby.

```
~$ rvm list known
```

Check Ruby in the *AWS Elastic Beanstalk Platforms* document to find the latest version of Ruby available on an Elastic Beanstalk platform. Install that version.

```
~$ rvm install 3.2
```

Test your Ruby installation.

```
~$ ruby --version
```

Installing the AWS SDK for Ruby

If you need to manage AWS resources from within your application, install the AWS SDK for Ruby. For example, with the SDK for Ruby, you can use Amazon DynamoDB (DynamoDB) to store user and session information without creating a relational database.

Install the SDK for Ruby and its dependencies with the gem command.

```
$ gem install aws-sdk
```

Visit the AWS SDK for Ruby homepage for more information and installation instructions.

Installing an IDE or text editor

Integrated development environments (IDEs) provide a wide range of features that facilitate application development. If you haven't used an IDE for Ruby development, try Aptana and RubyMine and see which works best for you.

- Install Aptana
- RubyMine

Note

An IDE might add files to your project folder that you might not want to commit to source control. To prevent committing these files to source control, use `.gitignore` or your source control tool's equivalent.

If you just want to begin coding and don't need all of the features of an IDE, consider installing Sublime Text.

Using the Elastic Beanstalk Ruby platform

This topic describes how to configure, build, and run your Ruby applications on Elastic Beanstalk.

AWS Elastic Beanstalk supports a number of platform branches for different versions of the Ruby programming language. See Ruby in the *AWS Elastic Beanstalk Platforms* document for a full list.

The Ruby web application can run behind an NGINX proxy server under a Puma application server. If you use RubyGems, you can include a `Gemfile` in your source bundle to install packages during deployment.

Application server configuration

Elastic Beanstalk installs the Puma application server based on the Ruby platform branch that you choose when you create your environment. For more information about the components provided with the Ruby platform versions, see Supported Platforms in the *AWS Elastic Beanstalk Platforms* guide.

You can configure your application with your own provided Puma server. This provides the option to use a version of Puma other than the one pre-installed with the Ruby platform branch. You can also configure your application to use a different application server, such as Passenger. To do so, you must include and customize a `Gemfile` in your deployment. You're also required to configure a `Procfile` to start the application server. For more information see *Configuring the application process with a Procfile*.

Other configuration options

Elastic Beanstalk provides configuration options that you can use to customize the software that runs on the Amazon Elastic Compute Cloud (Amazon EC2) instances in your Elastic Beanstalk environment. You can configure environment variables needed by your application, enable log rotation to Amazon S3, and map folders in your application source that contain static files to paths served by the proxy server. The platform also predefines some common environment variables related to Rails and Rack for ease of discovery and use.

Configuration options are available in the Elastic Beanstalk console for modifying the configuration of a running environment. To avoid losing your environment's configuration when you terminate it, you can use saved configurations to save your settings and later apply them to another environment.

To save settings in your source code, you can include configuration files. Settings in configuration files are applied every time you create an environment or deploy your application. You can also use configuration files to install packages, run scripts, and perform other instance customization operations during deployments.

Settings applied in the Elastic Beanstalk console override the same settings in configuration files, if they exist. This lets you have default settings in configuration files, and override them with environment-specific settings in the console. For more information about precedence, and other methods of changing settings, see Configuration options.

For details about the various ways you can extend an Elastic Beanstalk Linux-based platform, see the section called “Extending Linux platforms”.

Configuring your Ruby environment

You can use the Elastic Beanstalk console to enable log rotation to Amazon S3 and configure variables that your application can read from the environment.

To access the software configuration settings for your environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.

Log options

The **Log options** section has two settings:

- **Instance profile**– Specifies the instance profile that has permission to access the Amazon S3 bucket associated with your application.
- **Enable log file rotation to Amazon S3** – Specifies whether log files for your application's Amazon EC2 instances are copied to the Amazon S3 bucket associated with your application.

Static files

To improve performance, you can use the **Static files** section to configure the proxy server to serve static files (for example, HTML or images) from a set of directories inside your web application. For each directory, you set the virtual path to directory mapping. When the proxy server receives a request for a file under the specified path, it serves the file directly instead of routing the request to your application.

For details about configuring static files using configuration files or the Elastic Beanstalk console, see the section called "Static files".

By default, the proxy server in a Ruby environment is configured to serve static files as follows:

- Files in the `public` folder are served from the `/public` path and the domain root (/ path).
- Files in the `public/assets` subfolder are served from the `/assets` path.

The following examples illustrate how the default configuration works:

- If your application source contains a file named `logo.png` in a folder named `public`, the proxy server serves it to users from `*subdomain*.elasticbeanstalk.com/public/logo.png` and `*subdomain*.elasticbeanstalk.com/logo.png`.
- If your application source contains a file named `logo.png` in a folder named `assets` inside the `public` folder, the proxy server serves it from `*subdomain*.elasticbeanstalk.com/assets/logo.png`.

You can configure additional mappings for static files. For more information, see Ruby configuration namespaces later in this topic.

Note

For platform versions prior to *Ruby 2.7 AL2 version 3.3.7*, the default Elastic Beanstalk nginx proxy server configuration doesn't support serving static files from the domain root (*subdomain*.elasticbeanstalk.com/). This platform version was released on October 21, 2021. For more information see New platform versions - Ruby in the *AWS Elastic Beanstalk Release Notes*.

Environment properties

The **Environment Properties** section lets you specify environment configuration settings on the Amazon EC2 instances that are running your application. Environment properties are passed in as key-value pairs to the application.

The Ruby platform defines the following properties for environment configuration:

- **BUNDLE_WITHOUT** – A colon-separated list of groups to ignore when installing dependencies from a Gemfile.
- **BUNDLER_DEPLOYMENT_MODE** – Set to `true` (the default) to install dependencies in deployment mode using Bundler. Set to `false` to run `bundle install` in development mode.

  Note

  This environment property isn't defined on Amazon Linux AMI Ruby platform branches (preceding Amazon Linux 2).
- **RAILS_SKIP_ASSET_COMPILATION** – Set to `true` to skip running `rake assets:precompile` during deployment.
- **RAILS_SKIP_MIGRATIONS** – Set to `true` to skip running `rake db:migrate` during deployment.
- **RACK_ENV** – Specify the environment stage for Rack. For example, `development`, `production`, or `test`.

Inside the Ruby environment running in Elastic Beanstalk, environment variables are accessible using the `ENV` object. For example, you could read a property named `API_ENDPOINT` to a variable with the following code:

```
endpoint = ENV['API_ENDPOINT']
```

See Environment variables and other software settings for more information.

Ruby configuration namespaces

You can use a configuration file to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be platform specific or apply to all platforms in the Elastic Beanstalk service as a whole. Configuration options are organized into *namespaces*.

You can use the `aws:elasticbeanstalk:environment:proxy:staticfiles` namespace to configure the environment proxy to serve static files. You define mappings of virtual paths to application directories.

The Ruby platform doesn't define any platform-specific namespaces. Instead, it defines environment properties for common Rails and Rack options.

The following configuration file specifies a static files option that maps a directory named `staticimages` to the path `/images`, sets each of the platform defined environment properties, and sets an additional environment property named `LOGGING`.

Example .ebextensions/ruby-settings.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:staticfiles:
    /images: staticimages
  aws:elasticbeanstalk:application:environment:
    BUNDLE_WITHOUT: test
    BUNDLER_DEPLOYMENT_MODE: true
    RACK_ENV: development
    RAILS_SKIP_ASSET_COMPILATION: true
    RAILS_SKIP_MIGRATIONS: true
    LOGGING: debug
```

Note

The `BUNDLER_DEPLOYMENT_MODE` environment property and the `aws:elasticbeanstalk:environment:proxy:staticfiles` namespace aren't defined on Amazon Linux AMI Ruby platform branches (preceding Amazon Linux 2).

Elastic Beanstalk provides many configuration options for customizing your environment. In addition to configuration files, you can also set configuration options using the console, saved configurations, the EB CLI, or the AWS CLI. See Configuration options for more information.

Installing packages with a Gemfile on Elastic Beanstalk

To use RubyGems to install packages that your application requires, include a `Gemfile` file in the root of your project source.

Example Gemfile

```
source "https://rubygems.org"
gem 'sinatra'
gem 'json'
gem 'rack-parser'
```

When a `Gemfile` file is present, Elastic Beanstalk runs `bundle install` to install dependencies. For more information, see the Gemfiles and Bundle pages on the Bundler.io website.

Note

You can use a different version of Puma besides the default that's pre-installed with the Ruby platform. To do so, include an entry in a `Gemfile` that specifies the version. You can also specify a different application server, such as Passenger, by using a customized `Gemfile`.
For both of these cases you're required to configure a `Procfile` to start the application server.
For more information see *Configuring the application process with a Procfile*.

Configuring the application process with a Procfile on Elastic Beanstalk.

To specify the command that starts your Ruby application, include a file called `Procfile` at the root of your source bundle.

Note

Elastic Beanstalk doesn't support this feature on Amazon Linux AMI Ruby platform branches (preceding Amazon Linux 2). Platform branches with names containing *with Puma* or *with Passenger*, regardless of their Ruby versions, precede Amazon Linux 2 and don't support the `Procfile` feature.

For details about writing and using a `Procfile`, see Buildfile and Procfile.

When you don't provide a `Procfile`, Elastic Beanstalk generates a default `Procfile`. If your `Gemfile` includes Puma, Elastic Beanstalk assumes you want to use your provided version of Puma and generates the following default `Procfile`.

```
web: bundle exec puma -C /opt/elasticbeanstalk/config/private/pumaconf.rb
```

If your `Gemfile` does not include Puma, Elastic Beanstalk assumes you're using the pre-installed Puma application server and generates the following default `Procfile`. On Amazon Linux 2 Ruby platform branches, Elastic Beanstalk always generates the following default `Procfile` if you don't provide a `Procfile`.

```
web: puma -C /opt/elasticbeanstalk/config/private/pumaconf.rb
```

Note

On October 10, 2024, the last Ruby Amazon Linux 2 platform branches were retired. All currently supported Ruby platform branches are based on Amazon Linux 2023. For information about migration, see Migration from Amazon Linux 2 to Amazon Linux 2023.

If you want to use the Passenger application server, use the following example files to configure your Ruby environment to install and use Passenger.

1. Use this example file to install Passenger.

   Example Gemfile

   ```
   source 'https://rubygems.org'
   gem 'passenger'
   ```

2. Use this example file to instruct Elastic Beanstalk to start Passenger.

Example Procfile

```
web: bundle exec passenger start /var/app/current --socket /var/run/puma/my_app.sock
```

> **Note**
>
> You don't have to change anything in the configuration of the nginx proxy server to use Passenger. To use other application servers, you might need to customize the nginx configuration to properly forward requests to your application.

Deploying a rails application to Elastic Beanstalk

Rails is an open source, model-view-controller (MVC) framework for Ruby. This tutorial walks you through the process of generating a Rails application and deploying it to an AWS Elastic Beanstalk environment.

Sections

Prerequisites

Basic Elastic Beanstalk knowledge

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in Learn how to get started with Elastic Beanstalk to launch your first Elastic Beanstalk environment.

Command line

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

Rails dependencies

The Rails framework 6.1.4.1 has the following dependencies. Be sure you have all of them installed.

- **Ruby 2.5.0 or newer** – For installation instructions, see Setting up your Ruby development environment for Elastic Beanstalk.

  In this tutorial we use Ruby 3.0.2 and the corresponding Elastic Beanstalk platform version.
- **Node.js** – For installation instructions, see Installing Node.js via package manager.
- **Yarn** – For installation instructions, see Installation on the *Yarn* website.

Launch an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. Choose the **Ruby** platform and accept the default settings and sample code.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link: console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.

5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about 5 minutes and creates the following resources:

- **EC2 instance –** An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

  Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or NGINX as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group –** An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer –** An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group –** An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.
- **Auto Scaling group –** An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket –** A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms –** Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack –** Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name –** A domain name that routes to your web app in the form *`subdomain`.`region`.elasticbeanstalk.com*.

Domain security

To augment the security of your Elastic Beanstalk applications, the *elasticbeanstalk.com* domain is registered in the Public Suffix List (PSL).

If you ever need to set sensitive cookies in the default domain name for your Elastic Beanstalk applications, we recommend that you use cookies with a `__Host-` prefix for increased security. This practice defends your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Note

The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see Using Elastic Beanstalk with Amazon S3.

Install rails and generate a website

Install Rails and its dependencies with the `gem` command.

```
~$ gem install rails
Fetching: concurrent-ruby-1.1.9.gem
Successfully installed concurrent-ruby-1.1.9
Fetching: rack-2.2.3.gem
Successfully installed rack-2.2.3
...
```

Test your Rails installation.

```
~$ rails --version
Rails 6.1.4.1
```

Use `rails new` with the name of the application to create a new Rails project.

```
~$ rails new ~/eb-rails
```

Rails creates a directory with the name specified, generates all of the files needed to run a sample project locally, and then runs bundler to install all of the dependencies (Gems) defined in the project's Gemfile.

Note

This process installs the latest Puma version for the project. This version might be different from the version that Elastic Beanstalk provides on the Ruby platform version of your environment. To see the Puma versions provided by Elastic Beanstalk, see Ruby Platform History in the *AWS Elastic Beanstalk Platforms guide*. For more information about the latest Puma version, see the Puma.io website. If there's a mismatch between the two Puma versions, use one of the following options:

- Use the Puma version installed by the prior `rails new` command. In this case you must add a `Procfile` for the platform to use your own provided Puma server version. For more information, see Configuring the application process with a Procfile on Elastic Beanstalk..
- Update the Puma version to be consistent with the version pre-installed on your environment's Ruby platform version. To do so, modify the Puma version in the Gemfile located in the root of your project source directory. Then run `bundle update`. For more information see the bundle update page on the Bundler.io website.

Test your Rails installation by running the default project locally.

```
~$ cd eb-rails
~/eb-rails$ rails server
=> Booting Puma
=> Rails 6.1.4.1 application starting in development
=> Run `bin/rails server --help` for more startup options
Puma starting in single mode...
* Puma version: 5.5.2 (ruby 3.0.2-p107) ("Zawgyi")
*  Min threads: 5
*  Max threads: 5
*  Environment: development
*          PID: 77857
* Listening on http://127.0.0.1:3000
* Listening on http://[::1]:3000
Use Ctrl-C to stop
...
```

Open `http://localhost:3000` in a web browser to see the default project in action.

This page is only visible in development mode. Add some content to the front page of the application to support production deployment to Elastic Beanstalk. Use `rails generate` to create a controller, route, and view for your welcome page.

```
~/eb-rails$ rails generate controller WelcomePage welcome
      create  app/controllers/welcome_page_controller.rb
       route  get 'welcome_page/welcome'
      invoke  erb
      create    app/views/welcome_page
      create    app/views/welcome_page/welcome.html.erb
      invoke  test_unit
      create    test/controllers/welcome_page_controller_test.rb
      invoke  helper
      create    app/helpers/welcome_page_helper.rb
      invoke    test_unit
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/welcome_page.coffee
      invoke    scss
      create      app/assets/stylesheets/welcome_page.scss.
```

This gives you all you need to access the page at `/welcome_page/welcome`. Before you publish the changes, however, change the content in the view and add a route to make this page appear at the top level of the site.

Use a text editor to edit the content in `app/views/welcome_page/welcome.html.erb`. For this example, you'll use `cat` to simply overwrite the content of the existing file.

Example app/views/welcome_page/welcome.html.erb

```
<h1>Welcome!</h1>
<p>This is the front page of my first Rails application on Elastic Beanstalk.</p>
```

Finally, add the following route to `config/routes.rb`:

Example config/routes.rb

```
Rails.application.routes.draw do
  get 'welcome_page/welcome'
  root 'welcome_page#welcome'
```

This tells Rails to route requests to the root of the website to the welcome page controller's welcome method, which renders the content in the welcome view (`welcome.html.erb`).

In order for Elastic Beanstalk to successfully deploy the application on the Ruby platform, we need to update `Gemfile.lock`. Some dependencies of `Gemfile.lock` might be platform specific. Therefore, we need to add **`platform ruby`** to `Gemfile.lock` so that all required dependencies are installed with the deployment.

Example

```
~/eb-rails$ bundle lock --add-platform ruby
```

```
Fetching gem metadata from https://rubygems.org/...........
Resolving dependencies...
Writing lockfile to /Users/janedoe/EBDPT/RubyApps/eb-rails-doc-app/Gemfile.lock
```

Configure rails settings

Use the Elastic Beanstalk console to configure Rails with environment properties. Set the SECRET_KEY_BASE environment property to a string of up to 256 alphanumeric characters.

Rails uses this property to create keys. Therefore you should keep it a secret and not store it in source control in plain text. Instead, you provide it to Rails code on your environment through an environment property.

To configure environment variables in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. Scroll down to **Runtime environment variables**.
6. Select **Add environment variable**.
7. For **Source** select **Plain text**.

   Note

   The **Secrets Manager** and **SSM Parameter Store** values in the drop-down are for configuring environment variables as secrets to store sensitive data, such as credentials and API keys. For more information, see Using Elastic Beanstalk with AWS Secrets Manager and AWS Systems Manager Parameter Store.

8. Enter the **Environment variable name** and **Environment variable value** pairs.
9. If you need to add more variables repeat **Step 6** through **Step 8**.
10. To save the changes choose **Apply** at the bottom of the page.

Now you're ready to deploy the site to your environment.

Deploy your application

Create a source bundle containing the files created by Rails. The following command creates a source bundle named `rails-default.zip`.

```
~/eb-rails$ zip ../rails-default.zip -r * .[^.]*
```

Upload the source bundle to Elastic Beanstalk to deploy Rails to your environment.

To deploy a source bundle

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Cleanup

After you finish working with the demo code, you can terminate your environment. Elastic Beanstalk deletes all related AWS resources, such as Amazon EC2 instances, database instances, load balancers, security groups, and alarms.

Removing resources does not delete the Elastic Beanstalk application, so you can create new environments for your application at any time.

To terminate your Elastic Beanstalk environment from the console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

Next steps

For more information about Rails, visit rubyonrails.org.

As you continue to develop your application, you'll probably want a way to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The Elastic Beanstalk Command Line Interface (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

Finally, if you plan on using your application in a production environment, you will want to configure a custom domain name for your environment and enable HTTPS for secure connections.

Deploying a sinatra application to Elastic Beanstalk

This walkthrough shows how to deploy a simple Sinatra web application to AWS Elastic Beanstalk.

Prerequisites

This tutorial assumes you have knowledge of the basic Elastic Beanstalk operations and the Elastic Beanstalk console. If you haven't already, follow the instructions in Learn how to get started with Elastic Beanstalk to launch your first Elastic Beanstalk environment.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

Sinatra 2.1.0 requires Ruby 2.3.0 or newer. In this tutorial we use Ruby 3.0.2 and the corresponding Elastic Beanstalk platform version. Install Ruby by following the instructions at Setting up your Ruby development environment for Elastic Beanstalk.

Launch an Elastic Beanstalk environment

Use the Elastic Beanstalk console to create an Elastic Beanstalk environment. Choose the **Ruby** platform and accept the default settings and sample code.

To launch an environment (console)

1. Open the Elastic Beanstalk console using this preconfigured link: console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application.
3. For **Application code**, choose **Sample application**.
4. Choose **Review and launch**.
5. Review the available options. Choose the available option you want to use, and when you're ready, choose **Create app**.

Environment creation takes about 5 minutes and creates the following resources:

- **EC2 instance** – An Amazon Elastic Compute Cloud (Amazon EC2) virtual machine configured to run web apps on the platform that you choose.

  Each platform runs a specific set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination of these. Most platforms use either Apache or NGINX as a reverse proxy that sits in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic isn't allowed on other ports.
- **Load balancer** – An Elastic Load Balancing load balancer configured to distribute requests to the instances running your application. A load balancer also eliminates the need to expose your instances directly to the internet.
- **Load balancer security group** – An Amazon EC2 security group configured to allow inbound traffic on port 80. This resource lets HTTP traffic from the internet reach the load balancer. By default, traffic isn't allowed on other ports.

- **Auto Scaling group** – An Auto Scaling group configured to replace an instance if it is terminated or becomes unavailable.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and that are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name** – A domain name that routes to your web app in the form *subdomain*.*region*.*elasticbeanstalk.com*.

> **Domain security**
>
> To augment the security of your Elastic Beanstalk applications, the *elasticbeanstalk.com* domain is registered in the Public Suffix List (PSL).
> If you ever need to set sensitive cookies in the default domain name for your Elastic Beanstalk applications, we recommend that you use cookies with a `__Host-` prefix for increased security. This practice defends your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

All of these resources are managed by Elastic Beanstalk. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

> **Note**
>
> The Amazon S3 bucket that Elastic Beanstalk creates is shared between environments and is not deleted during environment termination. For more information, see Using Elastic Beanstalk with Amazon S3.

Write a basic sinatra website

To create and deploy a sinatra application

1. Create a configuration file named **config.ru** with the following contents.

   Example config.ru

   ```
   require './helloworld'
   run Sinatra::Application
   ```

2. Create a Ruby code file named **helloworld.rb** with the following contents.

   Example helloworld.rb

   ```
   require 'sinatra'
   get '/' do
     "Hello World!"
   end
   ```

3. Create a **Gemfile** with the following contents.

   Example Gemfile

   ```
   source 'https://rubygems.org'
   gem 'sinatra'
   gem 'puma'
   ```

4. Run bundle install to generate the `Gemfile.lock`

   Example

   ```
   ~/eb-sinatra$ bundle install
   Fetching gem metadata from https://rubygems.org/....
   Resolving dependencies...
   Using bundler 2.2.22
   Using rack 2.2.3
   ...
   ```

5. In order for Elastic Beanstalk to successfully deploy the application on the Ruby platform, we need to update `Gemfile.lock`. Some dependencies of `Gemfile.lock` might be platform

specific. Therefore, we need to add **platform ruby** to Gemfile.lock so that all required dependencies are installed with the deployment.

Example

```
~/eb-sinatra$ bundle lock --add-platform ruby
Fetching gem metadata from https://rubygems.org/....
Resolving dependencies...
Writing lockfile to /Users/janedoe/EBDPT/RubyApps/eb-sinatra/Gemfile.lock
```

6. Create a Procfile with the following contents.

   Example Procfile

```
web: bundle exec puma -C /opt/elasticbeanstalk/config/private/pumaconf.rb
```

Deploy your application

Create a source bundle containing the your source files. The following command creates a source bundle named sinatra-default.zip.

```
~/eb-sinatra$ zip ../sinatra-default.zip -r * .[^.]*
```

Upload the source bundle to Elastic Beanstalk to deploy Sinatra to your environment.

To deploy a source bundle

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. On the environment overview page, choose **Upload and deploy**.
4. Use the on-screen dialog box to upload the source bundle.
5. Choose **Deploy**.
6. When the deployment completes, you can choose the site URL to open your website in a new tab.

Cleanup

After you finish working with the demo code, you can terminate your environment. Elastic Beanstalk deletes all related AWS resources, such as Amazon EC2 instances, database instances, load balancers, security groups, and alarms.

Removing resources does not delete the Elastic Beanstalk application, so you can create new environments for your application at any time.

To terminate your Elastic Beanstalk environment from the console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

Next steps

For more information about Sinatra, visit sinatrarb.com.

As you continue to develop your application, you'll probably want a way to manage environments and deploy your application without manually creating a .zip file and uploading it to the Elastic Beanstalk console. The Elastic Beanstalk Command Line Interface (EB CLI) provides easy-to-use commands for creating, configuring, and deploying applications to Elastic Beanstalk environments from the command line.

Finally, if you plan on using your application in a production environment, you will want to configure a custom domain name for your environment and enable HTTPS for secure connections.

Adding an Amazon RDS DB instance to your Ruby Elastic Beanstalk environment

This topic provides instructions to create an Amazon RDS using the Elastic Beanstalk console. You can use an Amazon Relational Database Service (Amazon RDS) DB instance to store data gathered and modified by your application. The database can be coupled to your environment and managed by Elastic Beanstalk, or it can be created as decoupled and managed externally by another service. In these instructions the database is coupled to your environment and managed by Elastic Beanstalk. For more information about integrating an Amazon RDS with Elastic Beanstalk, see Adding a database to your Elastic Beanstalk environment.

Sections

Adding a DB instance to your environment

To add a DB instance to your environment

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Database** configuration category, choose **Edit**.
5. Choose a DB engine, and enter a user name and password.
6. To save the changes choose **Apply** at the bottom of the page.

Adding a DB instance takes about 10 minutes. When the environment update is complete, the DB instance's hostname and other connection information are available to your application through the following environment properties:

Property name	Description	Property value
`RDS_HOSTNAME`	The hostname of the DB instance.	On the **Connectivity & security** tab on the Amazon RDS console: **Endpoint**.
`RDS_PORT`	The port where the DB instance accepts connectio ns. The default value varies among DB engines.	On the **Connectivity & security** tab on the Amazon RDS console: **Port**.

Property name	Description	Property value
RDS_DB_NAME	The database name, **ebdb**.	On the **Configuration** tab on the Amazon RDS console: **DB Name**.
RDS_USERNAME	The username that you configured for your database.	On the **Configuration** tab on the Amazon RDS console: **Master username**.
RDS_PASSWORD	The password that you configured for your database.	Not available for reference in the Amazon RDS console.

For more information about configuring a database instance coupled with an Elastic Beanstalk environment, see Adding a database to your Elastic Beanstalk environment.

Downloading an adapter

Add the database adapter to your project's gem file.

Example Gemfile – Rails with MySQL

```
source 'https://rubygems.org'
gem 'puma'
gem 'rails', '~> 6.1.4', '>= 6.1.4.1'
gem 'mysql2'
```

Common adapter gems for Ruby

- **MySQL** – `mysql2`
- **PostgreSQL** – `pg`
- **Oracle** – `activerecord-oracle_enhanced-adapter`
- **SQL Server** – `activerecord-sqlserver-adapter`

Connecting to a database

Elastic Beanstalk provides connection information for attached DB instances in environment properties. Use `ENV['`*`VARIABLE`*`']` to read the properties and configure a database connection.

Example config/database.yml – Ruby on rails database configuration (MySQL)

```
production:
  adapter: mysql2
  encoding: utf8
  database: <%= ENV['RDS_DB_NAME'] %>
  username: <%= ENV['RDS_USERNAME'] %>
  password: <%= ENV['RDS_PASSWORD'] %>
  host: <%= ENV['RDS_HOSTNAME'] %>
  port: <%= ENV['RDS_PORT'] %>
```

Deploying with Docker containers to Elastic Beanstalk

This chapter explains how you can use Elastic Beanstalk to deploy web applications from Docker containers. Docker containers are self contained and include all the configuration information and software that your web application requires to run. With Docker containers you can define your own runtime environment. You can also choose your own programming language and application dependencies, such as package managers or tools, which typically aren't supported by other Elastic Beanstalk platforms.

Follow the steps in QuickStart for Docker to create a Docker "Hello World" application and deploy it to an Elastic Beanstalk environment using the EB CLI.

Topics

Elastic Beanstalk Docker platform branches

The Elastic Beanstalk Docker platform supports the following platform branches:

Docker running Amazon Linux 2* and *Docker running AL2023

Elastic Beanstalk deploys Docker container(s) and source code to EC2 instances and manages them. These platform branches offer multi-container support. You can use the Docker Compose tool to simplify your application configuration, testing, and deployment. For more information about this platform branch, see the section called "Docker platform branch".

ECS running on Amazon Linux 2* and *ECS running on AL2023

We provide this branch for customers who need a migration path to AL2023/AL2 from the retired platform branch *Multi-container Docker running on (Amazon Linux AMI)*. The latest platform branches support all of the features from the retired platform branch. No changes to the source code are required. For more information, see Migrating your Elastic Beanstalk application from ECS managed Multi-container Docker on AL1 to ECS on Amazon Linux 2023. If you don't have an Elastic Beanstalk environment running on an ECS based platform branch, we recommend you use the platform branch, *Docker Running on 64bit AL2023*. This offers a simpler approach and requires less resources.

For a list of the software component versions associated with each of these platform branches, see Docker in the *AWS Elastic Beanstalk Platforms* document.

Retired platform branches running on Amazon Linux AMI (AL1)

On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. Expand each section that follows to read more about each retired platform branch and its migration path to the latest platform branch running on Amazon Linux 2 or Amazon Linux 2023 (recommended).

Docker (Amazon Linux AMI)

This platform branch can deploy a Docker image, described in a Dockerfile or a `Dockerrun.aws.json` v1 definition. This platform branch runs *only one* container for each instance. Its succeeding platform branches,*Docker running on 64bit AL2023* and *Docker running on 64bit Amazon Linux 2* support multiple Docker containers per instance.

We recommend that you create your environments with the newer and supported platform branch *Docker running on 64bit AL2023*. You can then migrate your application to the newly created environment. For more information about creating these environments, see the section called "Docker platform branch". For more information about migration, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

Multi-container Docker (Amazon Linux AMI)

This platform branch uses Amazon ECS to coordinate a deployment of multiple Docker containers to an Amazon ECS cluster in an Elastic Beanstalk environment. If you're currently using this retired platform branch, we recommend that you migrate to the latest *ECS Running on Amazon Linux 2023* platform branch. The latest platform branch supports all of the features from this discontinued platform branch. No changes to the source code are required. For more information, see Migrating your Elastic Beanstalk application from ECS managed Multi-container Docker on AL1 to ECS on Amazon Linux 2023.

Preconfigured Docker containers

In addition to the prior mentioned Docker platforms, there is also the *Preconfigured Docker GlassFish* platform branch that runs on the Amazon Linux AMI operating system (AL1).

This platform branch has been superseded by the platform branches *Docker running on 64bit AL2023* and *Docker running on 64bit Amazon Linux 2*. For more information, see Deploying a GlassFish application to the Docker platform.

Using the Elastic Beanstalk Docker platform branch

This section describes how to prepare your Docker image for launch with the either of the Elastic Beanstalk platform branches *Docker running AL2 or AL2023*.

Follow the steps in QuickStart for Docker to create a Docker "Hello World" application and deploy it to an Elastic Beanstalk environment using the EB CLI.

Topics

QuickStart: Deploy a Docker application to Elastic Beanstalk

This QuickStart tutorial walks you through the process of creating a Docker application and deploying it to an AWS Elastic Beanstalk environment.

Note

Tutorial examples are intended for demonstration. Do not use the application for production traffic.

Sections

Your AWS account

If you're not already an AWS customer, you need to create an AWS account. Signing up enables you to access Elastic Beanstalk and other AWS services that you need.

If you already have an AWS account, you can move on to Prerequisites.

Create an AWS account

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.
2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.
2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Prerequisites

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

EB CLI

This tutorial uses the Elastic Beanstalk Command Line Interface (EB CLI). For details on installing and configuring the EB CLI, see Install EB CLI with setup script (recommended) and Configure the EB CLI.

Docker

To follow this tutorial, you'll need a working local installation of Docker. For more information, see Get Docker on the Docker documentation website.

Verify the Docker daemon is up an running by running the following command.

```
~$ docker info
```

Step 1: Create a Docker application and container

For this example, we create a Docker image of the sample Flask application that's also referenced in Deploying a Flask application to Elastic Beanstalk.

The application consists of two files:

- `app.py`— the Python file that contains the code that will execute in the container.
- `Dockerfile`— the Dockerfile to build your image.

Place both files at the root of a directory.

```
~/eb-docker-flask/
|-- Dockerfile
|-- app.py
```

Add the following contents to your `Dockerfile`.

Example ~/eb-docker-flask/Dockerfile

```
FROM public.ecr.aws/docker/library/python:3.12
COPY . /app
WORKDIR /app
RUN pip install Flask==3.1.1
EXPOSE 5000
CMD [ "python3", "-m" , "flask", "run", "--host=0.0.0.0"]
```

Add the following contents to your `app.py` file.

Example ~/eb-docker-flask/app.py

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello Elastic Beanstalk! This is a Docker application'
```

Use the docker build command to build your container image locally, tagging the image with `eb-docker-flask`. The period (`.`) at the end of the command specificies that path is a local directory.

```
~/eb-docker-flask$ docker build -t eb-docker-flask .
```

Step 2: Run your application locally

Run your container with the docker run command. The command will print the ID of the running container. The **-d** option runs docker in background mode. The **-p** option exposes your application at port 5000. Elastic Beanstalk serves traffic to port 5000 on the Docker platform by default.

```
~/eb-docker-flask$ docker run -dp 127.0.0.1:5000:5000 eb-docker-flask
```

Navigate to `http://127.0.0.1:5000/` in your browser. You should see the text "Hello Elastic Beanstalk! This is a Docker application".

Run the docker kill command to terminate the container.

```
~/eb-docker-flask$ docker kill container-id
```

Step 3: Deploy your Docker application with the EB CLI

Run the following commands to create an Elastic Beanstalk environment for this application.

To create an environment and deploy your Docker application

1. Initialize your EB CLI repository with the **eb init** command.

   ```
   ~/eb-docker-flask$ eb init -p docker docker-tutorial --region us-east-2
   Application docker-tutorial has been created.
   ```

   This command creates an application named `docker-tutorial` and configures your local repository to create environments with the latest Docker platform version.

2. (Optional) Run **eb init** again to configure a default key pair so that you can use SSH to connect to the EC2 instance running your application.

   ```
   ~/eb-docker-flask$ eb init
   Do you want to set up SSH for your instances?
   (y/n): y
   Select a keypair.
   1) my-keypair
   2) [ Create new KeyPair ]
   ```

Select a key pair if you have one already, or follow the prompts to create one. If you don't see the prompt or need to change your settings later, run **eb init -i**.

3. Create an environment and deploy your application to it with **eb create**. Elastic Beanstalk automatically builds a zip file for your application and starts it on port 5000.

   ```
   ~/eb-docker-flask$ eb create docker-tutorial
   ```

   It takes about five minutes for Elastic Beanstalk to create your environment.

Step 4: Run your application on Elastic Beanstalk

When the process to create your environment completes, open your website with **eb open**.

```
~/eb-docker-flask$ eb open
```

Congratulations! You've deployed a Docker application with Elastic Beanstalk! This opens a browser window using the domain name created for your application.

Step 5: Clean up

You can terminate your environment when you finish working with your application. Elastic Beanstalk terminates all AWS resources associated with your environment.

To terminate your Elastic Beanstalk environment with the EB CLI run the following command.

```
~/eb-docker-flask$ eb terminate
```

AWS resources for your application

You just created a single instance application. It serves as a straightforward sample application with a single EC2 instance, so it doesn't require load balancing or auto scaling. For single instance applications Elastic Beanstalk creates the following AWS resources:

- **EC2 instance –** An Amazon EC2 virtual machine configured to run web apps on the platform you choose.

  Each platform runs a different set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination thereof. Most platforms use either Apache or nginx as a reverse proxy that processes web traffic in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.

- **Instance security group** – An Amazon EC2 security group configured to allow incoming traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic is not allowed on other ports.
- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name** – A domain name that routes to your web app in the form *subdomain*.*region*.*elasticbeanstalk.com*.

Elastic Beanstalk manages all of these resources. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains.

Next steps

After you have an environment running an application, you can deploy a new version of the application or a different application at any time. Deploying a new application version is very quick because it doesn't require provisioning or restarting EC2 instances. You can also explore your new environment using the Elastic Beanstalk console. For detailed steps, see Explore your environment in the *Getting started* chapter of this guide.

After you deploy a sample application or two and are ready to start developing and running Docker applications locally, see Preparing your Docker image for deployment to Elastic Beanstalk.

Deploy with the Elastic Beanstalk console

You can also use the Elastic Beanstalk console to launch the sample application. For detailed steps, see Create an example application in the *Getting started* chapter of this guide.

QuickStart: Deploy a Docker Compose application to Elastic Beanstalk

This QuickStart tutorial walks you through the process of creating a multi-container Docker Compose application and deploying it to an AWS Elastic Beanstalk environment. You'll create a Flask web application with an nginx reverse proxy to demonstrate how Docker Compose simplifies orchestrating multiple containers.

Note

Tutorial examples are intended for demonstration. Do not use the application for production traffic.

Sections

Your AWS account

If you're not already an AWS customer, you need to create an AWS account. Signing up enables you to access Elastic Beanstalk and other AWS services that you need.

If you already have an AWS account, you can move on to Prerequisites.

Create an AWS account

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.
2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Prerequisites

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/eb-project$ this is a command
this is output
```

On Linux and macOS, you can use your preferred shell and package manager. On Windows you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

EB CLI

This tutorial uses the Elastic Beanstalk Command Line Interface (EB CLI). For details on installing and configuring the EB CLI, see Install EB CLI with setup script (recommended) and Configure the EB CLI.

Docker and Docker Compose

To follow this tutorial, you'll need a working local installation of Docker and Docker Compose. For more information, see Get Docker and Install Docker Compose on the Docker documentation website.

Verify that Docker and Docker Compose are installed and running by running the following commands.

```
~$ docker info
~$ docker compose version
```

Step 1: Create a Docker Compose application

For this example, we create a multi-container application using Docker Compose that consists of a Flask web application and an nginx reverse proxy. This demonstrates how Docker Compose simplifies orchestrating multiple containers that work together.

The application includes health monitoring configuration that allows Elastic Beanstalk to collect detailed application metrics from your nginx proxy.

The application consists of the following structure:

```
~/eb-docker-compose-flask/
|-- docker-compose.yml
|-- web/
|   |-- Dockerfile
|   |-- app.py
|   `-- requirements.txt
|-- proxy/
|   |-- Dockerfile
|   `-- nginx.conf
`-- .platform/
    `-- hooks/
        `-- postdeploy/
```

```
            `-- 01_setup_healthd_permissions.sh
```

Create the directory structure and add the following files:

First, create the main `docker-compose.yml` file that defines the services and their relationships.

Example ~/eb-docker-compose-flask/docker-compose.yml

```
services:
  web:
    build: ./web
    expose:
      - "5000"

  nginx-proxy:
    build: ./proxy
    ports:
      - "80:80"
    volumes:
      - "/var/log/nginx:/var/log/nginx"
    depends_on:
      - web
```

Create the Flask web application in the `web` directory. Add the following contents to your `app.py` file.

Example ~/eb-docker-compose-flask/web/app.py

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello Elastic Beanstalk! This is a Docker Compose application'
```

Add the following contents to your web service `Dockerfile`.

Example ~/eb-docker-compose-flask/web/Dockerfile

```
FROM public.ecr.aws/docker/library/python:3.12
COPY . /app
WORKDIR /app
RUN pip install Flask==3.1.1
```

```
EXPOSE 5000
CMD [ "python3", "-m" , "flask", "run", "--host=0.0.0.0"]
```

Create the nginx reverse proxy in the `proxy` directory. Add the following contents to your `nginx.conf` file.

This configuration includes health monitoring setup that allows Elastic Beanstalk to collect detailed application metrics. For more information about customizing health monitoring log formats, see Enhanced health log format.

Example ~/eb-docker-compose-flask/proxy/nginx.conf

```
events {
    worker_connections 1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    map $http_upgrade $connection_upgrade {
        default       "upgrade";
    }

    # Health monitoring log format for Elastic Beanstalk
    log_format healthd
 '$msec"$uri"$status"$request_time"$upstream_response_time"$http_x_forwarded_for';

    upstream flask_app {
        server web:5000;
    }

    server {
        listen 80 default_server;
        root /usr/share/nginx/html;

        # Standard access log
        access_log /var/log/nginx/access.log;

        # Health monitoring log for Elastic Beanstalk
        if ($time_iso8601 ~ "^(\d{4})-(\d{2})-(\d{2})T(\d{2})") {
            set $year $1;
            set $month $2;
            set $day $3;
            set $hour $4;
        }
        access_log /var/log/nginx/healthd/application.log.$year-$month-$day-$hour
 healthd;

        location / {
            proxy_pass http://flask_app;
            proxy_http_version    1.1;

            proxy_set_header    Connection             $connection_upgrade;
            proxy_set_header    Upgrade                $http_upgrade;
            proxy_set_header    Host                   $host;
            proxy_set_header    X-Real-IP              $remote_addr;
            proxy_set_header    X-Forwarded-For        $proxy_add_x_forwarded_for;
        }
    }
}
```

Add the following contents to your proxy service `Dockerfile`.

Example ~/eb-docker-compose-flask/proxy/Dockerfile

```
FROM public.ecr.aws/nginx/nginx:alpine
COPY nginx.conf /etc/nginx/nginx.conf
EXPOSE 80
```

Finally, create a platform hook script to set up the necessary log directories and permissions for health monitoring. Platform hooks allow you to run custom scripts during the deployment process. For more information about platform hooks, see Platform hooks.

Example ~/eb-docker-compose-flask/.platform/hooks/postdeploy/01_setup_healthd_permissions.sh

```
#!/bin/bash
set -ex

NGINX_CONTAINER=$(docker ps --filter "name=nginx-proxy" -q)

if [ -z "$NGINX_CONTAINER" ]; then
    echo "Error: No nginx-proxy container found running"
    exit 1
```

```
fi

NGINX_UID=$(docker exec ${NGINX_CONTAINER} id -u nginx)
NGINX_GID=$(docker exec ${NGINX_CONTAINER} id -g nginx)

mkdir -p /var/log/nginx/healthd
chown -R ${NGINX_UID}:${NGINX_GID} /var/log/nginx
```

Step 2: Run your application locally

Use the docker compose up command to build and run your multi-container application locally. Docker Compose will build both container images and start the services defined in your `docker-compose.yml` file.

```
~/eb-docker-compose-flask$ docker compose up --build
```

The **--build** option ensures that Docker Compose builds the container images before starting the services. You should see output showing both the web service and nginx-proxy service starting up.

Navigate to `http://localhost` in your browser. You should see the text "Hello Elastic Beanstalk! This is a Docker Compose application". The nginx proxy receives your request on port 80 and forwards it to the Flask application running on port 5000.

When you're finished testing, stop the application by pressing **Ctrl+C** in the terminal, or run the following command in a separate terminal:

```
~/eb-docker-compose-flask$ docker compose down
```

Step 3: Deploy your Docker Compose application with the EB CLI

Run the following commands to create an Elastic Beanstalk environment for this application.

To create an environment and deploy your Docker Compose application

1. Initialize your EB CLI repository with the **eb init** command.

   ```
   ~/eb-docker-compose-flask$ eb init -p docker docker-compose-tutorial --region us-
   east-2
   Application docker-compose-tutorial has been created.
   ```

This command creates an application named `docker-compose-tutorial` and configures your local repository to create environments with the latest Docker platform version.

2. (Optional) Run **eb init** again to configure a default key pair so that you can use SSH to connect to the EC2 instance running your application.

   ```
   ~/eb-docker-compose-flask$ eb init
   Do you want to set up SSH for your instances?
   (y/n): y
   Select a keypair.
   1) my-keypair
   2) [ Create new KeyPair ]
   ```

   Select a key pair if you have one already, or follow the prompts to create one. If you don't see the prompt or need to change your settings later, run **eb init -i**.

3. Create an environment and deploy your application to it with **eb create**. Elastic Beanstalk automatically detects your `docker-compose.yml` file and deploys your multi-container application.

   ```
   ~/eb-docker-compose-flask$ eb create docker-compose-tutorial
   ```

   It takes about five minutes for Elastic Beanstalk to create your environment and deploy your multi-container application.

Step 4: Test your application on Elastic Beanstalk

When the process to create your environment completes, open your website with **eb open**.

```
~/eb-docker-compose-flask$ eb open
```

Great! You've deployed a multi-container Docker Compose application with Elastic Beanstalk! This opens a browser window using the domain name created for your application. You should see the message from your Flask application, served through the nginx reverse proxy.

Step 5: Clean up

You can terminate your environment when you finish working with your application. Elastic Beanstalk terminates all AWS resources associated with your environment.

To terminate your Elastic Beanstalk environment with the EB CLI run the following command.

```
~/eb-docker-compose-flask$ eb terminate
```

AWS resources for your application

You just created a single instance application running multiple containers. It serves as a straightforward sample application with a single EC2 instance, so it doesn't require load balancing or auto scaling. For single instance applications Elastic Beanstalk creates the following AWS resources:

- **EC2 instance –** An Amazon EC2 virtual machine configured to run web apps on the platform you choose.

  Each platform runs a different set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination thereof. Most platforms use either Apache or nginx as a reverse proxy that processes web traffic in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group –** An Amazon EC2 security group configured to allow incoming traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic is not allowed on other ports.
- **Amazon S3 bucket –** A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms –** Two CloudWatch alarms that monitor the load on the instances in your environment and are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack –** Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name –** A domain name that routes to your web app in the form *`subdomain`.`region`.elasticbeanstalk.com*.

Elastic Beanstalk manages all of these resources. When you terminate your environment, Elastic Beanstalk terminates all the resources that it contains. Your Docker Compose application runs multiple containers on the single EC2 instance, with Elastic Beanstalk handling the orchestration automatically.

Next steps

After you have an environment running an application, you can deploy a new version of the application or a different application at any time. Deploying a new application version is very quick because it doesn't require provisioning or restarting EC2 instances. You can also explore your new environment using the Elastic Beanstalk console. For detailed steps, see Explore your environment in the *Getting started* chapter of this guide.

After you deploy a sample application or two and are ready to start developing and running Docker Compose applications locally, see Preparing your Docker image for deployment to Elastic Beanstalk.

Deploy with the Elastic Beanstalk console

You can also use the Elastic Beanstalk console to launch a Docker Compose application. Create a ZIP file containing your `docker-compose.yml` file and all associated directories and files, then upload it when creating a new application. For detailed steps, see Create an example application in the *Getting started* chapter of this guide.

Preparing your Docker image for deployment to Elastic Beanstalk

This section describes how to prepare your Docker image for deployment to Elastic Beanstalk with either of the *Docker running AL2 or AL2023* platform branches. The configuration files that you'll require depend on whether your images are local, remote, and if you're using Docker Compose.

Note

For an example of a procedure that launches a Docker environment see the QuickStart for Docker topic.

Topics

- Managing your images with Docker Compose in Elastic Beanstalk
- Managing images without Docker Compose in Elastic Beanstalk
- Building custom images with a Dockerfile

Managing your images with Docker Compose in Elastic Beanstalk

You may choose to use Docker Compose to manage various services in one YAML file. To learn more about Docker Compose see Why use Compose? on the Docker website.

- Create a `docker-compose.yml`. This file is required if you're using Docker Compose to manage your application with Elastic Beanstalk. If all your deployments are sourced from images in public repositories, then no other configuration files are required. If your deployment's source images are in a private repository, you'll need to do some additional configuration. For more information, see Using images from a private repository. For more information about the `docker-compose.yml` file, see Compose file reference on the Docker website.
- The `Dockerfile` is optional. Create one if you need Elastic Beanstalk to build and run a local custom image. For more information about the `Dockerfile` see Dockerfile reference on the Docker website.
- You may need to create a `.zip` file. If you use only a `Dockerfile` file to deploy your application, you don't need to create one. If you use additional configuration files the .zip file must include the `Dockerfile`, the `docker-compose.yml` file, your application files, and any application file dependencies. The `Dockerfile` and the `docker-compose.yml` must be at the root, or top level, of the .zip archive. If you use the EB CLI to deploy your application, it creates a .zip file for you.

To learn more about Docker Compose and how to install it, see the Docker sites Overview of Docker Compose and Install Docker Compose.

Managing images without Docker Compose in Elastic Beanstalk

If you're not using Docker Compose to manage your Docker images, you'll need to configure a `Dockerfile`, a `Dockerrun.aws.json` file, or both.

- Create a `Dockerfile` to have Elastic Beanstalk build and run a custom image locally.
- Create a `Dockerrun.aws.json v1` file to deploy a Docker image from a hosted repository to Elastic Beanstalk.
- You may need to create a `.zip` file. If you use *only one* of either file, the `Dockerfile` or the `Dockerrun.aws.json`, then you don't need to create a .zip file. If you use both files, then you do need a .zip file. The .zip file must include both the `Dockerfile` and the `Dockerrun.aws.json`, along with the file containing your application files plus any application file dependencies. If you use the EB CLI to deploy your application, it creates a `.zip` file for you.

`Dockerrun.aws.json` v1 configuration file

A `Dockerrun.aws.json` file describes how to deploy a remote Docker image as an Elastic Beanstalk application. This JSON file is specific to Elastic Beanstalk. If your application runs on an image that is available in a hosted repository, you can specify the image in a `Dockerrun.aws.json v1` file and omit the `Dockerfile`.

`Dockerrun.aws.json` versions

The `AWSEBDockerrunVersion` parameter indicates the version of the `Dockerrun.aws.json` file.

- The Docker AL2 and AL2023 platforms use the following versions of the file.
  - `Dockerrun.aws.json v3` — environments that use Docker Compose.
  - `Dockerrun.aws.json v1` — environments that do not use Docker Compose.
- *ECS running on Amazon Linux 2* and *ECS running on AL2023* uses the `Dockerrun.aws.json v2` file. The retired platform *ECS-The Multicontainer Docker Amazon Linux AMI (AL1)* also used this same version.

Dockerrun.aws.json v1

Valid keys and values for the `Dockerrun.aws.json v1` file include the following operations:

AWSEBDockerrunVersion

(Required) Specify the version number 1 if you're not using Docker Compose to manage your image.

Authentication

(Required only for private repositories) Specifies the Amazon S3 object storing the `.dockercfg` file.

See Authenticating with image repositories in *Using images from a private repository* later in this chapter.

Image

Specifies the Docker base image on an existing Docker repository from which you're building a Docker container. Specify the value of the **Name** key in the format *`<organization>/<image`*

name> for images on Docker Hub, or *<site>/<organization name>/<image name>* for other sites.

When you specify an image in the `Dockerrun.aws.json` file, each instance in your Elastic Beanstalk environment runs `docker pull` to run the image. Optionally, include the **Update** key. The default value is `true` and instructs Elastic Beanstalk to check the repository, pull any updates to the image, and overwrite any cached images.

When using a `Dockerfile`, do not specify the **Image** key in the `Dockerrun.aws.json` file. Elastic Beanstalk always builds and uses the image described in the `Dockerfile` when one is present.

Ports

(Required when you specify the **Image** key) Lists the ports to expose on the Docker container. Elastic Beanstalk uses the **ContainerPort** value to connect the Docker container to the reverse proxy running on the host.

You can specify multiple container ports, but Elastic Beanstalk uses only the first port. It uses this port to connect your container to the host's reverse proxy and route requests from the public internet. If you're using a `Dockerfile`, the first **ContainerPort** value should match the first entry in the `Dockerfile`'s **EXPOSE** list.

Optionally, you can specify a list of ports in **HostPort**. **HostPort** entries specify the host ports that **ContainerPort** values are mapped to. If you don't specify a **HostPort** value, it defaults to the **ContainerPort** value.

```
{
  "Image": {
    "Name": "image-name"
  },
  "Ports": [
    {
      "ContainerPort": 8080,
      "HostPort": 8000
    }
  ]
}
```

Volumes

Map volumes from an EC2 instance to your Docker container. Specify one or more arrays of volumes to map.

```
{
  "Volumes": [
    {
      "HostDirectory": "/path/inside/host",
      "ContainerDirectory": "/path/inside/container"
    }
  ]
...
```

Logging

Specify the directory inside the container to which your application writes logs. Elastic Beanstalk uploads any logs in this directory to Amazon S3 when you request tail or bundle logs. If you rotate logs to a folder named `rotated` within this directory, you can also configure Elastic Beanstalk to upload rotated logs to Amazon S3 for permanent storage. For more information, see Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment.

Command

Specify a command to run in the container. If you specify an **Entrypoint**, then **Command** is added as an argument to **Entrypoint**. For more information, see CMD in the Docker documentation.

Entrypoint

Specify a default command to run when the container starts. For more information, see ENTRYPOINT in the Docker documentation.

The following snippet is an example that illustrates the syntax of the `Dockerrun.aws.json` file for a single container.

```
{
  "AWSEBDockerrunVersion": "1",
  "Image": {
    "Name": "janedoe/image",
    "Update": "true"
  },
  "Ports": [
    {
      "ContainerPort": "1234"
    }
  ],
  "Volumes": [
    {
      "HostDirectory": "/var/app/mydb",
      "ContainerDirectory": "/etc/mysql"
    }
  ],
  "Logging": "/var/log/nginx",
  "Entrypoint": "/app/bin/myapp",
  "Command": "--argument"
}>
```

You can provide Elastic Beanstalk with only the `Dockerrun.aws.json` file, or with a `.zip` archive containing both the `Dockerrun.aws.json` and `Dockerfile` files. When you provide both files, the `Dockerfile` describes the Docker image and the `Dockerrun.aws.json` file provides additional information for deployment, as described later in this section.

Note

The two files must be at the root, or top level, of the `.zip` archive. Don't build the archive from a directory containing the files. Instead, navigate into that directory and build the archive there.
When you provide both files, don't specify an image in the `Dockerrun.aws.json` file. Elastic Beanstalk builds and uses the image described in the `Dockerfile` and ignores the image specified in the `Dockerrun.aws.json` file.

Building custom images with a Dockerfile

You need to create a `Dockerfile` if you don't already have an existing image hosted in a repository.

The following snippet is an example of the `Dockerfile`. If you follow the instructions in QuickStart for Docker, you can upload this `Dockerfile` as written. Elastic Beanstalk runs the game 2048 when you use this `Dockerfile`.

For more information about instructions you can include in the `Dockerfile`, see Dockerfile reference on the Docker website.

```
FROM ubuntu:12.04

RUN apt-get update
RUN apt-get install -y nginx zip curl

RUN echo "daemon off;" >> /etc/nginx/nginx.conf
RUN curl -o /usr/share/nginx/www/master.zip -L https://codeload.github.com/
gabrielecirulli/2048/zip/master
RUN cd /usr/share/nginx/www/ && unzip master.zip && mv 2048-master/* . && rm -rf 2048-
master master.zip

EXPOSE 80

CMD ["/usr/sbin/nginx", "-c", "/etc/nginx/nginx.conf"]
```

Note

You can run multi-stage builds from a single Dockerfile to produce smaller-sized images with a significant reduction in complexity. For more information, see Use multi-stage builds on the Docker documentation website.

Using the ECS managed Docker platform branch in Elastic Beanstalk

This topic provides an overview of the Elastic Beanstalk ECS managed Docker platform branches for Amazon Linux 2 and Amazon Linux 2023. It also provides configuration information that's specific to the Docker ECS managed platform.

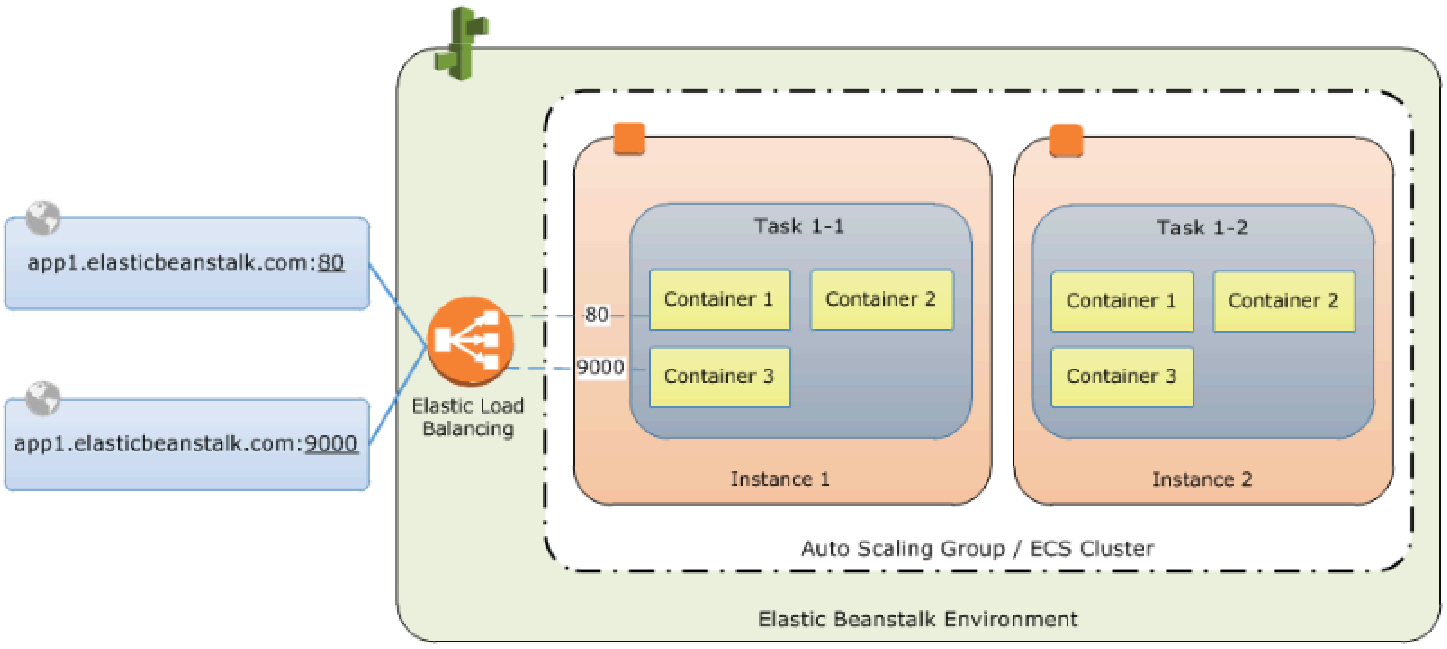
Migration from Multi-container Docker on AL1

On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. Although this chapter provides configuration information for this retired platform, we strongly recommend that you migrate to the latest supported platform branch. If you're presently using the retired *Multi-container Docker running on AL1* platform branch, you can migrate to the latest *ECS Running on AL2023* platform branch. The latest platform branch supports all of the features from the discontinued platform branch. No changes to the source code are required. For more information, see Migrating your Elastic Beanstalk application from ECS managed Multi-container Docker on AL1 to ECS on Amazon Linux 2023.

ECS managed Docker platform overview

Elastic Beanstalk uses Amazon Elastic Container Service (Amazon ECS) to coordinate container deployments to ECS managed Docker environments. Amazon ECS provides tools to manage a cluster of instances running Docker containers. Elastic Beanstalk takes care of Amazon ECS tasks including cluster creation, task definition and execution. Each of the instances in the environment run the same set of containers, which are defined in a `Dockerrun.aws.json` v2 file. In order to get the most out of Docker, Elastic Beanstalk lets you create an environment where your Amazon EC2 instances run multiple Docker containers side by side.

The following diagram shows an example Elastic Beanstalk environment configured with three Docker containers running on each Amazon EC2 instance in an Auto Scaling group:



Amazon ECS resources created by Elastic Beanstalk

When you create an environment using the ECS managed Docker platform, Elastic Beanstalk automatically creates and configures several Amazon Elastic Container Service resources while building the environment. In doing so, it creates the necessary containers on each Amazon EC2 instance.

- **Amazon ECS Cluster** – Container instances in Amazon ECS are organized into clusters. When used with Elastic Beanstalk, one cluster is always created for each ECS managed Docker environment. An ECS cluster also contains Auto Scaling group capacity providers and other resources.
- **Amazon ECS Task Definition** – Elastic Beanstalk uses the `Dockerrun.aws.json` v2 in your project to generate the Amazon ECS task definition that is used to configure container instances in the environment.
- **Amazon ECS Task** – Elastic Beanstalk communicates with Amazon ECS to run a task on every instance in the environment to coordinate container deployment. In a scalable environment, Elastic Beanstalk initiates a new task whenever an instance is added to the cluster.
- **Amazon ECS Container Agent** – The agent runs in a Docker container on the instances in your environment. The agent polls the Amazon ECS service and waits for a task to run.
- **Amazon ECS Data Volumes** – In addition to the volumes that you define in the `Dockerrun.aws.json` v2, Elastic Beanstalk inserts volume definitions into the task definition to facilitate log collection.

  Elastic Beanstalk creates log volumes on the container instance, one for each container, at `/var/log/containers/`*`containername`*. These volumes are named `awseb-logs-`*`containername`* and are provided for containers to mount. See Container definition format for details on how to mount them.

For more information about Amazon ECS resources, see the Amazon Elastic Container Service Developer Guide.

`Dockerrun.aws.json` v2 file

The container instances require a configuration file named `Dockerrun.aws.json`. *Container instances* refers to Amazon EC2 instances running ECS managed Docker in an Elastic Beanstalk environment. This file is specific to Elastic Beanstalk and can be used alone or combined with source code and content in a source bundle to create an environment on a Docker platform.

Note

The Version 2 format of the `Dockerrun.aws.json` adds support for multiple containers per Amazon EC2 instance and can only be used with an ECS managed Docker platform.

The format differs significantly from the other configuration file versions that support the Docker platform branches that aren't managed by ECS.

See the `Dockerrun.aws.json` v2 for details on the updated format and an example file.

Docker images

The ECS managed Docker platform for Elastic Beanstalk requires images to be prebuilt and stored in a public or private online image repository before creating an Elastic Beanstalk environment.

Note

Building custom images during deployment with a `Dockerfile` is not supported by the ECS managed Docker platform on Elastic Beanstalk. Build your images and deploy them to an online repository before creating an Elastic Beanstalk environment.

Specify images by name in `Dockerrun.aws.json` v2.

To configure Elastic Beanstalk to authenticate to a private repository, include the `authentication` parameter in your `Dockerrun.aws.json` v2 file.

Failed container deployments

If an Amazon ECS task fails, one or more containers in your Elastic Beanstalk environment will not start. Elastic Beanstalk does not roll back multi-container environments due to a failed Amazon ECS task. If a container fails to start in your environment, redeploy the current version or a previous working version from the Elastic Beanstalk console.

To deploy an existing version

1. Open the Elastic Beanstalk console in your environment's region.
2. Click **Actions** to the right of your application name and then click **View application versions**.
3. Select a version of your application and click **Deploy**.

Extending ECS based Docker platforms for Elastic Beanstalk

Elastic Beanstalk offers extensibility features that enable you to apply your own commands, scripts, software, and configurations to your application deployments. The deployment workflow for the *ECS AL2 and AL2023* platform branches varies slightly from the other Linux based platforms. For more information, see Instance deployment workflow for ECS running on Amazon Linux 2 and later.

ECS managed Docker configuration for Elastic Beanstalk

This chapter explains how to configure your ECS managed Docker environment. The following list summarizes the configuration items that this chapter explains.

- `Dockerrun.aws.json` v2 – This configuration file specifies your image repository and the name of your Docker images, among other components.
- EC2 Instance profile role – If you have a custom instance profile, we explain how to configure it so that the permissions required for ECS to manage your containers stay current.
- Elastic Load Balancing listeners – You'll need to configure multiple Elastic Load Balancing listeners if you need your environment to support inbound traffic for proxies or other services that don't run on the default HTTP port.

Topics

Configuring the Dockerrun.aws.json v2 file

`Dockerrun.aws.json v2` is an Elastic Beanstalk configuration file that describes how to deploy a set of Docker containers hosted in an ECS cluster in an Elastic Beanstalk environment. The Elastic Beanstalk platform creates an ECS *task definition*, which includes an ECS *container definition*. These definitions are described in the `Dockerrun.aws.json` configuration file.

The container definition in the `Dockerrun.aws.json` file describes the containers to deploy to each Amazon EC2 instance in the ECS cluster. In this case an Amazon EC2 instance is also referred to as a host *container instance*, because it hosts the Docker containers. The configuration file also describes the data volumes to create on the host container instance for the Docker containers to mount. For more information and a diagram of the components in an ECS managed Docker environment on Elastic Beanstalk, see the ECS managed Docker platform overview earlier in this chapter.

A `Dockerrun.aws.json` file can be used on its own or zipped up with additional source code in a single archive. Source code that is archived with a `Dockerrun.aws.json` is deployed to Amazon EC2 container instances and accessible in the `/var/app/current/` directory.

Topics

`Dockerrun.aws.json` v2

The `Dockerrun.aws.json` file includes the following sections:

AWSEBDockerrunVersion

Specifies the version number as the value 2 for ECS managed Docker environments.

executionRoleArn

Specifies the task execution IAM roles for different purposes and services associated with your account. For your application to use Elastic Beanstalk environment variables stored as secrets, you'll need to specify the ARN of a task execution role that grants the required permissions. Other common use cases may also require this parameter. For more information, see Execution Role ARN format.

volumes

Creates volumes from folders in the Amazon EC2 container instance, or from your source bundle (deployed to `/var/app/current`). Mount these volumes to paths within your Docker containers using `mountPoints` in the `containerDefinitions` section.

containerDefinitions

An array of container definitions.

authentication (optional)

The location in Amazon S3 of a `.dockercfg` file that contains authentication data for a private repository.

The *containerDefinitions* and *volumes* sections of `Dockerrun.aws.json` use the same formatting as the corresponding sections of an Amazon ECS task definition file. For more information about the task definition format and a full list of task definition parameters, see Amazon ECS task definitions in the *Amazon Elastic Container Service Developer Guide*.

Volume format

The *volume* parameter creates volumes from either folders in the Amazon EC2 container instance, or from your source bundle (deployed to `/var/app/current`).

Volumes are specified in the following format:

```
"volumes": [
    {
      "name": "volumename",
      "host": {
        "sourcePath": "/path/on/host/instance"
      }
    }
  ],
```

Mount these volumes to paths within your Docker containers using `mountPoints` in the container definition.

Elastic Beanstalk configures additional volumes for logs, one for each container. These should be mounted by your Docker containers in order to write logs to the host instance.

For more details, see the `mountPoints` field in the *Container definition format* section that follows.

Execution Role ARN format

In order for your application to use Elastic Beanstalk environment variables stored as secrets, you'll need to specify a task execution IAM role. The role must grant the Amazon ECS container permission to make AWS API calls on your behalf using AWS Secrets Manager secrets or AWS Systems Manager Parameter Store parameters to reference sensitive data. For instructions to create a task execution IAM role with the required permissions for your account, see Amazon ECS task execution IAM role in the Amazon Elastic Container Service Developer Guide.

```
{
"AWSEBDockerrunVersion": 2,
  "executionRoleArn": "arn:aws:iam::111122223333:role/ecsTaskExecutionRole",
```

Additional permissions required for the Amazon ECS managed Docker platform

EC2 instance profile grants `iam:PassRole` to ECS

In order for your EC2 instance profile to be able to grant this role to the ECS container, you must include the `iam:PassRole` permission demonstrated in the following example. The `iam:PassRole` allows the EC2 instances permission *to pass* the task execution role to the ECS container.

In this example, we limit the EC2 instance to only pass the role to the ECS service. Although this condition is not required, we add it to follow best practices to reduce the scope of the permission shared. We accomplish this with the `Condition` element.

Note

Any usage of the ECS IAM task execution role requires the `iam:PassRole` permission. There are other common use cases that require the ECS task execution managed service role. For more information, see Amazon ECS task execution IAM role in the Amazon Elastic Container Service Developer Guide.

Example policy with `iam:PassRole` permission

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws:iam::123456789012:role/ecs-task-execution-role"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": "ecs-tasks.amazonaws.com"
                }
            }
        }
    ]
}
```

Granting secrets and parameters access to the Amazon ECS container agent

The Amazon ECS task execution IAM role also needs permissions to access the secrets and parameter stores. Similar to the requirements of the EC2 instance profile role, the ECS container agent requires permission to pull the necessary Secrets Manager or Systems Manager resources. For more information, see Secrets Manager or Systems Manager permissions in the *Amazon Elastic Container Service Developer Guide*

Granting secrets and parameters access to the Elastic Beanstalk EC2 instances

To support secrets configured as environment variables, you'll also need to add permissions to your EC2 instance profile. For more information, see Fetching secrets and parameters to Elastic Beanstalk environment variables and Required IAM permissions for Secrets Manager.

The following examples combine the previous `iam:PassRole` example with the examples provided in the referenced Required IAM permissions for Secrets Manager. They add the permissions that the EC2 instances require to access the AWS Secrets Manager and AWS Systems Manager stores to retrieve the secrets and parameter data to initialize the Elastic Beanstalk environment variables that have been configured for secrets.

Example Secrets Manager policy combined with `iam:PassRole` permission

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
```

```
                "Effect": "Allow",
                "Action": "iam:PassRole",
                "Resource": [
                    "arn:aws:iam::123456789012:role/ecs-task-execution-role"
                ],
                "Condition": {
                    "StringLike": {
                        "iam:PassedToService": "ecs-tasks.amazonaws.com"
                   }
                }
            },
            {
                "Effect": "Allow",
                "Action": [
                    "secretsmanager:GetSecretValue",
                    "kms:Decrypt"
                ],
                "Resource": [
                    "arn:aws:secretsmanager:us-east-1:111122223333:secret:my-secret",
                    "arn:aws:kms:us-east-1:111122223333:key/my-key"
                ]
            }
        ]
}
```

Example Systems Manager policy combined with `iam:PassRole` permission

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
                "Effect": "Allow",
                "Action": "iam:PassRole",
                "Resource": [
                    "arn:aws:iam::123456789012:role/ecs-task-execution-role"
                ],
                "Condition": {
                    "StringLike": {
                        "iam:PassedToService": "ecs-tasks.amazonaws.com"
                   }
```

```
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameter",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:ssm:us-east-1:111122223333:parameter/my-parameter",
                "arn:aws:kms:us-east-1:111122223333:key/my-key"
            ]
        }
    ]
}
```

Container definition format

The following examples show a subset of parameters that are commonly used in the *containerDefinitions* section. More optional parameters are available.

The Beanstalk platform creates an ECS *task definition*, which includes an ECS *container definition*. Beanstalk supports a sub-set of parameters for the ECS container definition. For more information, see Container definitions in the *Amazon Elastic Container Service Developer Guide*.

A `Dockerrun.aws.json` file contains an array of one or more container definition objects with the following fields:

name

The name of the container. See Standard Container Definition Parameters for information about the maximum length and allowed characters.

image

The name of a Docker image in an online Docker repository from which you're building a Docker container. Note these conventions:

- Images in official repositories on Docker Hub use a single name (for example, `ubuntu` or `mongo`).
- Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`.

- Images in other online repositories are qualified further by a domain name (for example, `quay.io/assemblyline/ubuntu`).

environment

An array of environment variables to pass to the container.

For example, the following entry defines an environment variable with the name **`Container`** and the value **PHP**:

```
"environment": [
  {
    "name": "Container",
    "value": "PHP"
  }
],
```

essential

True if the task should stop if the container fails. Nonessential containers can finish or crash without affecting the rest of the containers on the instance.

memory

Amount of memory on the container instance to reserve for the container. Specify a non-zero integer for one or both of the `memory` or `memoryReservation` parameters in container definitions.

memoryReservation

The soft limit (in MiB) of memory to reserve for the container. Specify a non-zero integer for one or both of the `memory` or `memoryReservation` parameters in container definitions.

mountPoints

Volumes from the Amazon EC2 container instance to mount, and the location on the Docker container file system at which to mount them. When you mount volumes that contain application content, your container can read the data you upload in your source bundle. When you mount log volumes for writing log data, Elastic Beanstalk can gather log data from these volumes.

Elastic Beanstalk creates log volumes on the container instance, one for each Docker container, at `/var/log/containers/`*`containername`*. These volumes are named `awseb-`

logs-*containername* and should be mounted to the location within the container file structure where logs are written.

For example, the following mount point maps the nginx log location in the container to the Elastic Beanstalk–generated volume for the nginx-proxy container.

```
{
  "sourceVolume": "awseb-logs-nginx-proxy",
  "containerPath": "/var/log/nginx"
}
```

portMappings

Maps network ports on the container to ports on the host.

links

List of containers to link to. Linked containers can discover each other and communicate securely.

volumesFrom

Mount all of the volumes from a different container. For example, to mount volumes from a container named web:

```
"volumesFrom": [
  {
    "sourceContainer": "web"
  }
],
```

Authentication format – using images from a private repository

The authentication section contains authentication data for a private repository. This entry is optional.

Add the information about the Amazon S3 bucket that contains the authentication file in the authentication parameter of the Dockerrun.aws.json file. Make sure that the authentication parameter contains a valid Amazon S3 bucket and key. The Amazon S3 bucket must be hosted in the same region as the environment that is using it. Elastic Beanstalk will not download files from Amazon S3 buckets hosted in other regions.

Uses the following format:

```
"authentication": {
    "bucket": "amzn-s3-demo-bucket",
    "key": "mydockercfg"
  },
```

For information about generating and uploading the authentication file, see Authenticating with image repositories.

Example Dockerrun.aws.json v2

The following snippet is an example that illustrates the syntax of the `Dockerrun.aws.json` file for an instance with two containers.

```
{
  "AWSEBDockerrunVersion": 2,
  "volumes": [
    {
      "name": "php-app",
      "host": {
        "sourcePath": "/var/app/current/php-app"
      }
    },
    {
      "name": "nginx-proxy-conf",
      "host": {
        "sourcePath": "/var/app/current/proxy/conf.d"
      }
    }
  ],
  "containerDefinitions": [
    {
      "name": "php-app",
      "image": "php:fpm",
      "environment": [
        {
          "name": "Container",
          "value": "PHP"
        }
      ],
      "essential": true,
      "memory": 128,
      "mountPoints": [
        {
          "sourceVolume": "php-app",
          "containerPath": "/var/www/html",
          "readOnly": true
        }
      ]
    },
    {
      "name": "nginx-proxy",
      "image": "nginx",
      "essential": true,
      "memory": 128,
      "portMappings": [
        {
          "hostPort": 80,
          "containerPort": 80
        }
      ],
      "links": [
        "php-app"
      ],
      "mountPoints": [
        {
          "sourceVolume": "php-app",
          "containerPath": "/var/www/html",
          "readOnly": true
        },
        {
          "sourceVolume": "nginx-proxy-conf",
          "containerPath": "/etc/nginx/conf.d",
          "readOnly": true
        },
        {
          "sourceVolume": "awseb-logs-nginx-proxy",
          "containerPath": "/var/log/nginx"
        }
      ]
    }
  ]
}
```

Container managed policy and EC2 instance role

When you create an environment in the Elastic Beanstalk console, it prompts you to create a default instance profile that includes the `AWSElasticBeanstalkMulticontainerDocker` managed policy. So initially, your default EC2 instance profile, should include this managed policy. If your environment uses a custom EC2 instance profile role instead of the default, make sure that the managed policy `AWSElasticBeanstalkMulticontainerDocker` is attached so the required permissions for container management stay up-to-date.

Elastic Beanstalk uses an Amazon ECS-optimized AMI with an Amazon ECS container agent that runs in a Docker container. The agent communicates with Amazon ECS to coordinate container deployments. In order to communicate with Amazon ECS, each Amazon EC2 instance must have the corresponding IAM permissions, which are specified in this managed policy. See the AWSElasticBeanstalkMulticontainerDocker in the *AWS Managed Policy Reference Guide* to view these permissions.

If you use Elastic Beanstalk environment variables that are configured to access secrets or parameters that are stored in AWS Secrets Manager or AWS Systems Manager Parameter Store, you must customize your EC2 instance profile with additional permissions. For more information, see Execution Role ARN format.

Using multiple Elastic Load Balancing listeners

You can configure multiple Elastic Load Balancing listeners on a ECS managed Docker environment in order to support inbound traffic for proxies or other services that don't run on the default HTTP port.

Create a `.ebextensions` folder in your source bundle and add a file with a `.config` file extension. The following example shows a configuration file that creates an Elastic Load Balancing listener on port 8080.

`.ebextensions/elb-listener.config`

```
option_settings:
  aws:elb:listener:8080:
    ListenerProtocol: HTTP
    InstanceProtocol: HTTP
    InstancePort: 8080
```

If your environment is running in a custom Amazon Virtual Private Cloud (Amazon VPC) that you created, Elastic Beanstalk takes care of the rest. In a default VPC, you need to configure your instance's security group to allow ingress from the load balancer. Add a second configuration file that adds an ingress rule to the security group:

.ebextensions/elb-ingress.config

```
Resources:
  port8080SecurityGroupIngress:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      GroupId: {"Fn::GetAtt" : ["AWSEBSecurityGroup", "GroupId"]}
      IpProtocol: tcp
      ToPort: 8080
      FromPort: 8080
      SourceSecurityGroupName: { "Fn::GetAtt": ["AWSEBLoadBalancer",
 "SourceSecurityGroup.GroupName"] }
```

For more information on the configuration file format, see Adding and customizing Elastic Beanstalk environment resources and Option settings.

In addition to adding a listener to the Elastic Load Balancing configuration and opening a port in the security group, you need to map the port on the host instance to a port on the Docker container in the `containerDefinitions` section of the `Dockerrun.aws.json` v2 file. The following excerpt shows an example:

```
"portMappings": [
  {
    "hostPort": 8080,
    "containerPort": 8080
  }
]
```

See `Dockerrun.aws.json` v2 for details about the `Dockerrun.aws.json` v2 file format.

Creating an ECS managed Docker environment with the Elastic Beanstalk console

This tutorial details container configuration and source code preparation for an ECS managed Docker environment that uses two containers.

The containers, a PHP application and an nginx proxy, run side by side on each of the Amazon Elastic Compute Cloud (Amazon EC2) instances in an Elastic Beanstalk environment. After creating the environment and verifying that the applications are running, you'll connect to a container instance to see how it all fits together.

Sections

Define ECS managed Docker containers

The first step in creating a new Docker environment is to create a directory for your application data. This folder can be located anywhere on your local machine and have any name you choose. In addition to a container configuration file, this folder will contain the content that you will upload to Elastic Beanstalk and deploy to your environment.

Note

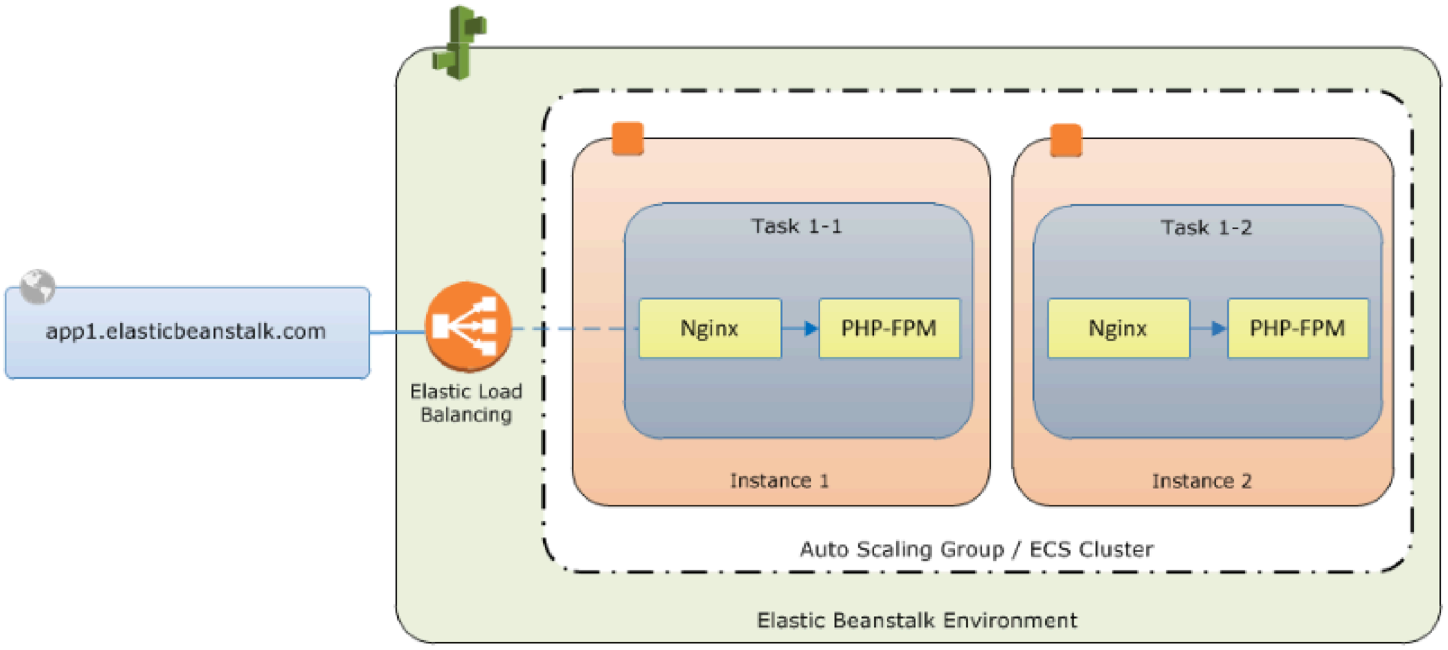
All of the code for this tutorial is available in the awslabs repository on GitHub at https://github.com/awslabs/eb-docker-nginx-proxy.

The file that Elastic Beanstalk uses to configure the containers on an Amazon EC2 instance is a JSON-formatted text file named `Dockerrun.aws.json` v2. The ECS managed Docker platform versions use a Version 2 format of this file. This format can only be used with the ECS managed Docker platform, as it differs significantly from the other configuration file versions that support the Docker platform branches that aren't managed by ECS.

Create a `Dockerrun.aws.json` v2 text file with this name at the root of your application and add the following text:

```
{
  "AWSEBDockerrunVersion": 2,
  "volumes": [
    {
      "name": "php-app",
      "host": {
        "sourcePath": "/var/app/current/php-app"
      }
    },
    {
      "name": "nginx-proxy-conf",
      "host": {
        "sourcePath": "/var/app/current/proxy/conf.d"
      }
    }
  ],
  "containerDefinitions": [
    {
      "name": "php-app",
      "image": "php:fpm",
      "essential": true,
      "memory": 128,
      "mountPoints": [
        {
          "sourceVolume": "php-app",
          "containerPath": "/var/www/html",
          "readOnly": true
        }
      ]
    },
    {
      "name": "nginx-proxy",
      "image": "nginx",
      "essential": true,
      "memory": 128,
      "portMappings": [
        {
          "hostPort": 80,
          "containerPort": 80
        }
      ],
      "links": [
        "php-app"
      ],
      "mountPoints": [
        {
          "sourceVolume": "php-app",
          "containerPath": "/var/www/html",
          "readOnly": true
        },
        {
          "sourceVolume": "nginx-proxy-conf",
          "containerPath": "/etc/nginx/conf.d",
          "readOnly": true
        },
        {
          "sourceVolume": "awseb-logs-nginx-proxy",
          "containerPath": "/var/log/nginx"
        }
      ]
    }
  ]
}
```

This example configuration defines two containers, a PHP web site with an nginx proxy in front of it. These two containers will run side by side in Docker containers on each instance in your Elastic Beanstalk environment, accessing shared content (the content of the website) from volumes on the host instance, which are also defined in this file. The containers themselves are created from images hosted in official repositories on Docker Hub. The resulting environment looks like the following:


app1.elasticbeanstalk.com
Elastic Load Balancing
Task 1-1
Nginx
PHP-FPM
Instance 1
Task 1-2
Nginx
PHP-FPM
Instance 2
Auto Scaling Group / ECS Cluster
Elastic Beanstalk Environment

The volumes defined in the configuration correspond to the content that you will create next and upload as part of your application source bundle. The containers access content on the host by mounting volumes in the `mountPoints` section of the container definitions.

For more information on the format of `Dockerrun.aws.json` v2 and its parameters, see Container definition format.

Add content

Next you will add some content for your PHP site to display to visitors, and a configuration file for the nginx proxy.

php-app/index.php

```
<h1>Hello World!!!</h1>
<h3>PHP Version <pre><?= phpversion()?></pre></h3>
```

php-app/static.html

```
<h1>Hello World!</h1>
<h3>This is a static HTML page.</h3>
```

proxy/conf.d/default.conf

```
server {
  listen 80;
  server_name localhost;
  root /var/www/html;

  index index.php;

  location ~ [^/]\.php(/|$) {
    fastcgi_split_path_info ^(.+?\.php)(/.*)$;
    if (!-f $document_root$fastcgi_script_name) {
      return 404;
    }

    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_param PATH_INFO $fastcgi_path_info;
    fastcgi_param PATH_TRANSLATED $document_root$fastcgi_path_info;

    fastcgi_pass php-app:9000;
    fastcgi_index index.php;
  }
}
```

Deploy to Elastic Beanstalk

Your application folder now contains the following files:

```
### Dockerrun.aws.json
### php-app
#   ### index.php
#   ### static.html
### proxy
    ### conf.d
        ### default.conf
```

This is all you need to create the Elastic Beanstalk environment. Create a `.zip` archive of the above files and folders (not including the top-level project folder). To create the archive in Windows explorer, select the contents of the project folder, right-click, select **Send To**, and then click **Compressed (zipped) Folder**

Note

For information on the required file structure and instructions for creating archives in other environments, see Create an Elastic Beanstalk application source bundle

Next, upload the source bundle to Elastic Beanstalk and create your environment. For **Platform**, select **Docker**. For **Platform branch**, select **ECS running on 64bit Amazon Linux 2023**.

To launch an environment (console)

1. Open the Elastic Beanstalk console with this preconfigured link: console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
2. For **Platform**, select the platform and platform branch that match the language used by your application, or the Docker platform for container-based applications.
3. For **Application code**, choose **Upload your code**.
4. Choose **Local file**, choose **Choose file**, and then open the source bundle.
5. Choose **Review and launch**.
6. Review the available settings, and then choose **Create app**.

The Elastic Beanstalk console redirects you to the management dashboard for your new environment. This screen shows the health status of the environment and events output by the Elastic Beanstalk service. When the status is Green, click the URL next to the environment name to see your new website.

Connect to a container instance

Next you will connect to an Amazon EC2 instance in your Elastic Beanstalk environment to see some of the moving parts in action.

The easiest way to connect to an instance in your environment is by using the EB CLI. To use it, install the EB CLI, if you haven't done so already. You'll also need to configure your environment with an Amazon EC2 SSH key pair. Use either the console's security configuration page or the EB CLI eb init command to do that. To connect to an environment instance, use the EB CLI eb ssh command.

Now that your connected to an Amazon EC2 instance hosting your docker containers, you can see how things are set up. Run `ls` on `/var/app/current`:

```
[ec2-user@ip-10-0-0-117 ~]$ ls /var/app/current
Dockerrun.aws.json  php-app  proxy
```

This directory contains the files from the source bundle that you uploaded to Elastic Beanstalk during environment creation.

```
[ec2-user@ip-10-0-0-117 ~]$ ls /var/log/containers
nginx-proxy    nginx-proxy-4ba868dbb7f3-stdouterr.log
php-app        php-app-dcc3b3c8522c-stdouterr.log      rotated
```

This is where logs are created on the container instance and collected by Elastic Beanstalk. Elastic Beanstalk creates a volume in this directory for each container, which you mount to the container location where logs are written.

You can also take a look at Docker to see the running containers with `docker ps`.

```
[ec2-user@ip-10-0-0-117 ~]$ sudo docker ps
CONTAINER ID   IMAGE                                  COMMAND                          CREATED
  STATUS                   PORTS                                      NAMES
```

```
4ba868dbb7f3   nginx                          "/docker-entrypoint.…"   4 minutes ago
   Up 4 minutes          0.0.0.0:80->80/tcp, :::80->80/tcp   ecs-awseb-Tutorials-env-
dc2aywfjwg-1-nginx-proxy-acca84ef87c4aca15400
dcc3b3c8522c   php:fpm                        "docker-php-entrypoi…"   4 minutes ago
   Up 4 minutes          9000/tcp                            ecs-awseb-Tutorials-env-
dc2aywfjwg-1-php-app-b8d38ae288b7b09e8101
d9367c0baad6   amazon/amazon-ecs-agent:latest   "/agent"                 5 minutes ago
  Up 5 minutes (healthy)                                    ecs-agent
```

This shows the two running containers that you deployed, as well as the Amazon ECS container agent that coordinated the deployment.

Inspect the Amazon ECS container agent

Amazon EC2 instances in a ECS managed Docker environment on Elastic Beanstalk run an agent process in a Docker container. This agent connects to the Amazon ECS service in order to coordinate container deployments. These deployments run as tasks in Amazon ECS, which are configured in task definition files. Elastic Beanstalk creates these task definition files based on the `Dockerrun.aws.json` that you upload in a source bundle.

Check the status of the container agent with an HTTP get request to `http://localhost:51678/v1/metadata`:

```
[ec2-user@ip-10-0-0-117 ~]$ curl http://localhost:51678/v1/metadata
{
  "Cluster":"awseb-Tutorials-env-dc2aywfjwg",
  "ContainerInstanceArn":"arn:aws:ecs:us-west-2:123456789012:container-instance/awseb-
Tutorials-env-dc2aywfjwg/db7be5215cd74658aacfcb292a6b944f",
  "Version":"Amazon ECS Agent - v1.57.1 (089b7b64)"
}
```

This structure shows the name of the Amazon ECS cluster, and the ARN (Amazon Resource Name) of the cluster instance (the Amazon EC2 instance that you are connected to).

For more information, make an HTTP get request to `http://localhost:51678/v1/tasks`:

```
[ec2-user@ip-10-0-0-117 ~]$ curl http://localhost:51678/v1/tasks
{
   "Tasks":[
      {
         "Arn":"arn:aws:ecs:us-west-2:123456789012:task/awseb-Tutorials-env-dc2aywfjwg/
bbde7ebe1d4e4537ab1336340150a6d6",
         "DesiredStatus":"RUNNING",
         "KnownStatus":"RUNNING",
         "Family":"awseb-Tutorials-env-dc2aywfjwg",
         "Version":"1",
         "Containers":[
            {

 "DockerId":"dcc3b3c8522cb9510b7359689163814c0f1453b36b237204a3fd7a0b445d2ea6",
               "DockerName":"ecs-awseb-Tutorials-env-dc2aywfjwg-1-php-app-
b8d38ae288b7b09e8101",
               "Name":"php-app",
               "Volumes":[
                  {
                     "Source":"/var/app/current/php-app",
                     "Destination":"/var/www/html"
                  }
               ]
            },
            {

 "DockerId":"4ba868dbb7f3fb3328b8afeb2cb6cf03e3cb1cdd5b109e470f767d50b2c3e303",
               "DockerName":"ecs-awseb-Tutorials-env-dc2aywfjwg-1-nginx-proxy-
acca84ef87c4aca15400",
               "Name":"nginx-proxy",
               "Ports":[
                  {
                     "ContainerPort":80,
                     "Protocol":"tcp",
                     "HostPort":80
                  },
                  {
                     "ContainerPort":80,
                     "Protocol":"tcp",
                     "HostPort":80
                  }
               ],
               "Volumes":[
                  {
                     "Source":"/var/app/current/php-app",
                     "Destination":"/var/www/html"
                  },
                  {
                     "Source":"/var/log/containers/nginx-proxy",
                     "Destination":"/var/log/nginx"
```

```
                    },
                    {
                       "Source":"/var/app/current/proxy/conf.d",
                       "Destination":"/etc/nginx/conf.d"
                    }
                 ]
              }
           ]
        }
     ]
  }
```

This structure describes the task that is run to deploy the two docker containers from this tutorial's example project. The following information is displayed:

- **KnownStatus –** The `RUNNING` status indicates that the containers are still active.
- **Family –** The name of the task definition that Elastic Beanstalk created from `Dockerrun.aws.json`.
- **Version –** The version of the task definition. This is incremented each time the task definition file is updated.
- **Containers –** Information about the containers running on the instance.

Even more information is available from the Amazon ECS service itself, which you can call using the AWS Command Line Interface. For instructions on using the AWS CLI with Amazon ECS, and information about Amazon ECS in general, see the Amazon ECS User Guide.

Migrating your Elastic Beanstalk application from ECS managed Multi-container Docker on AL1 to ECS on Amazon Linux 2023

Note

On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**..

This topic guides you in the migration of your applications from the retired platform branch *Multi-container Docker running on 64bit Amazon Linux* to *ECS Running on 64bit AL2023*. This target platform branch is current and supported. Like the previous *Multi-container Docker AL1* branch, the newer *ECS AL2023* platform branch uses Amazon ECS to coordinate deployment of multiple Docker containers to an Amazon ECS cluster in an Elastic Beanstalk environment. The new *ECS AL2023* platform branch supports all of the features in the previous *Multi-container Docker AL1* platform branch. Also, the same `Dockerrun.aws.json` v2 file is supported.

Sections

Migrate with the Elastic Beanstalk console

To migrate using the Elastic Beanstalk console deploy the same source code to a new environment that's based on the *ECS Running on AL2023* platform branch. No changes to the source code are required.

To migrate to the *ECS Running on Amazon Linux 2023* platform branch

1. Using the application source that's already deployed to the old environment, create an application source bundle. You can use the same application source bundle and the same `Dockerrun.aws.json` v2 file.
2. Create a new environment using the *ECS Running on Amazon Linux 2023* platform branch. Use the source bundle from the prior step for **Application code**. For more detailed steps, see Deploy to Elastic Beanstalk in the *ECS managed Docker tutorial* earlier in this chapter.

Migrate with the AWS CLI

You also have the option to use the AWS Command Line Interface (AWS CLI) to migrate your existing *Multi-container Docker Amazon Linux Docker* environment to the newer *ECS AL2023* platform branch. In this case you don't need to create a new environment or redeploy your source code. You only need to run the AWS CLI update-environment command. It will perform a platform update to migrate your existing environment to the *ECS Amazon Linux 2023* platform branch.

Use the following syntax to migrate your environment to the new platform branch.

```
aws elasticbeanstalk update-environment \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 version running ECS" \
--region my-region
```

The following is an example of the command to migrate environment *beta-101* to *version 3.0.0* of the *ECS Amazon Linux 2023* platform branch in the *us-east-1* region.

```
aws elasticbeanstalk update-environment \
--environment-name beta-101 \
--solution-stack-name "64bit Amazon Linux 2023 v4.0.0 running ECS" \
--region us-east-1
```

The `solution-stack-name` parameter provides the platform branch and its version. Use the most recent platform branch *version* by specifying the proper *solution stack name*. The version of every platform branch is included in the *solution stack name*, as shown in the above example. For a list of the most current solution stacks for the Docker platform, see Supported platforms in the *AWS Elastic Beanstalk Platforms* guide.

Note

The list-available-solution-stacks command provides a list of the platform versions available for your account in an AWS Region.

```
aws elasticbeanstalk list-available-solution-stacks --region us-east-1 --query
 SolutionStacks
```

To learn more about the AWS CLI, see the *AWS Command Line Interface User Guide*. For more information about AWS CLI commands for Elastic Beanstalk, see the *AWS CLI Command Reference for Elastic Beanstalk*.

Authenticating with image repositories

This topic describes how to authenticate to online image repositories with Elastic Beanstalk. For private repositories, Elastic Beanstalk must authenticate before it can pull and deploy your images. For Amazon ECR Public, authentication is optional but provides higher rate limits and improved reliability.

Using images from an Amazon ECR repository

You can store your custom Docker images in AWS with Amazon Elastic Container Registry (Amazon ECR).

When you store your Docker images in Amazon ECR, Elastic Beanstalk automatically authenticates to the Amazon ECR registry with your environment's instance profile. Therefore you'll need to provide your instances with permission to access the images in your Amazon ECR repository. To do so add permissions to your environment's instance profile by attaching the AmazonEC2ContainerRegistryReadOnly managed policy to the instance profile. This provides read-only access to all the Amazon ECR repositories in your account. You also have the option to only access to single repository by using the following template to create a custom policy:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEbAuth",
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "AllowPull",
            "Effect": "Allow",
            "Resource": [
                "arn:aws:ecr:us-east-2:111122223333:repository/repository-name"
            ],
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetRepositoryPolicy",
                "ecr:DescribeRepositories",
                "ecr:ListImages",
                "ecr:BatchGetImage"
            ]
        }
    ]
}
```

Replace the Amazon Resource Name (ARN) in the above policy with the ARN of your repository.

You'll need to specify the image information in your `Dockerrun.aws.json` file. The configuration will be different depending on which platform you use.

For the ECS managed Docker platform, use the `image` key in a container definition object :

```
"containerDefinitions": [
        {
        "name": "my-image",
        "image": "account-id.dkr.ecr.us-east-2.amazonaws.com/repository-name:latest",
```

For the Docker platform refer to the image by URL. The URL goes in the `Image` definition of your `Dockerrun.aws.json` file:

```
  "Image": {
      "Name": "account-id.dkr.ecr.us-east-2.amazonaws.com/repository-name:latest",
      "Update": "true"
    },
```

Using AWS Systems Manager (SSM) Parameter Store or AWS Secrets Manager

Configure Elastic Beanstalk to authenticate with your private repository before deployment to enable access to your container images.

This approach uses the *prebuild* phase of the Elastic Beanstalk deployment process with two components:

- ebextensions to define environment variables that store repository credentials
- platform hook scripts to execute **docker login** before pulling images

The hook scripts securely retrieve credentials from environment variables that are populated from AWS Systems Manager Parameter Store or AWS Secrets Manager. This feature requires Elastic Beanstalk Docker and ECS managed Docker platforms released on or after March 26, 2025. For more details, see environment variable configuration.

To configure Elastic Beanstalk to authenticate to your private repository with AWS Systems Manager Parameter Store or AWS Secrets Manager

Note

Before proceeding, ensure you have set up your credentials in AWS Systems Manager Parameter Store or AWS Secrets Manager and configured the necessary IAM permissions. See Prerequisites to configure secrets as environment variables for details.

1. Create the following directory structure for your project:

```
### .ebextensions
#   ### env.config
### .platform
#   ### confighooks
#   #   ### prebuild
#   #       ### 01login.sh
#   ### hooks
#       ### prebuild
#           ### 01login.sh
### Dockerfile
```

2. Use AWS Systems Manager Parameter Store or AWS Secrets Manager to save the credentials of your private repository. This example shows both AWS Systems Manager Parameter Store and AWS Secrets Manager but you can choose to use just one of these services.

```
aws ssm put-parameter --name USER --type SecureString --value "username"
aws secretsmanager create-secret --name PASSWD --secret-string "passwd"
```

3. Create the following `env.config` file and place it in the `.ebextensions` directory as shown in the preceding directory structure. This configuration uses the aws:elasticbeanstalk:application:environmentsecrets namespace to initialize the USER and PASSWD Elastic Beanstalk environment variables to the values that are stored in the Systems Manager Parameter Store.

Note

Ensure the variable names USER and PASSWD match the parameter names used in the put-parameter and create-secret commands.

```
option_settings:
  aws:elasticbeanstalk:application:environmentsecrets:
    USER: arn:aws:ssm:us-east-1:111122223333:parameter/user
    PASSWD: arn:aws:secretsmanager:us-east-1:111122223333:passwd
```

4. Create the following `01login.sh` script file and place it in the following directories (also shown in the preceding directory structure):

   - `.platform/confighooks/prebuild`
   - `.platform/hooks/prebuild`

```
### example 01login.sh
#!/bin/bash
echo $PASSWD | docker login -u $USER --password-stdin
```

   The `01login.sh` script uses the environment variables configured in **Step 3** and securely passes the password to **docker login** via `stdin`. For more information about Docker authentication, see docker login in the Docker documentation.

   Notes

   - Hook files can be either binary files or script files starting with a **#!** line containing their interpreter path, such as **#!/bin/bash**.
   - For more information, see the section called "Platform hooks" in *Extending Elastic Beanstalk Linux platforms*.

Once authentication is configured, Elastic Beanstalk can pull and deploy images from your private repository.

Using the `Dockerrun.aws.json` file

This section describes another approach to authenticate Elastic Beanstalk to a private repository. With this approach, you generate an authentication file with the Docker command, and then upload the authentication file to an Amazon S3 bucket. You must also include the bucket information in your `Dockerrun.aws.json` file.

To generate and provide an authentication file to Elastic Beanstalk

1. Generate an authentication file with the **docker login** command. For repositories on Docker Hub, run **docker login**:

   ```
   $ docker login
   ```

   For other registries, include the URL of the registry server:

   ```
   $ docker login registry-server-url
   ```

   Note

   If your Elastic Beanstalk environment uses the Amazon Linux AMI Docker platform version (precedes Amazon Linux 2), read the relevant information in the section called "Docker configuration on Amazon Linux AMI (preceding Amazon Linux 2)".

   For more information about the authentication file, see Store images on Docker Hub and docker login on the Docker website.

2. Upload a copy of the authentication file that is named `.dockercfg` to a secure Amazon S3 bucket.

   - The Amazon S3 bucket must be hosted in the same AWS Region as the environment that is using it. Elastic Beanstalk cannot download files from an Amazon S3 bucket hosted in other Regions.
   - Grant permissions for the `s3:GetObject` operation to the IAM role in the instance profile. For more information, see Managing Elastic Beanstalk instance profiles.

3. Include the Amazon S3 bucket information in the `Authentication` parameter in your `Dockerrun.aws.json` file.

The following example shows the use of an authentication file named `mydockercfg` in a bucket named `amzn-s3-demo-bucket` to use a private image in a third-party registry. For the correct version number for `AWSEBDockerrunVersion`, see the note that follows the example.

```
{
  "AWSEBDockerrunVersion": "version-no",
  "Authentication": {
    "Bucket": "amzn-s3-demo-bucket",
    "Key": "mydockercfg"
  },
  "Image": {
    "Name": "quay.io/johndoe/private-image",
    "Update": "true"
  },
  "Ports": [
    {
      "ContainerPort": "1234"
    }
  ],
  "Volumes": [
    {
      "HostDirectory": "/var/app/mydb",
      "ContainerDirectory": "/etc/mysql"
    }
  ],
  "Logging": "/var/log/nginx"
}
```

`Dockerrun.aws.json` versions

The `AWSEBDockerrunVersion` parameter indicates the version of the `Dockerrun.aws.json` file.

- The Docker AL2 and AL2023 platforms use the following versions of the file.
  - `Dockerrun.aws.json v3` — environments that use Docker Compose.
  - `Dockerrun.aws.json v1` — environments that do not use Docker Compose.

- *ECS running on Amazon Linux 2* and *ECS running on AL2023* uses the `Dockerrun.aws.json v2` file. The retired platform *ECS-The Multicontainer Docker Amazon Linux AMI (AL1)* also used this same version.

After Elastic Beanstalk can authenticate with the online registry that hosts the private repository, your images can be deployed and pulled.

Using images from Amazon ECR Public

Amazon ECR Public is a public container registry that hosts Docker images. While Amazon ECR Public repositories are publicly accessible, authenticating provides higher rate limits and better reliability for your deployments.

Note

Amazon ECR Public authentication is not supported in China regions (`cn-*`) and AWS GovCloud regions (`us-gov-*`). In these regions, Elastic Beanstalk will use unauthenticated pulls.

To enable Amazon ECR Public authentication, add the following permissions to your environment's instance profile. For more information about Amazon ECR Public authentication, see Registry authentication in Amazon ECR public in the *Amazon Elastic Container Registry Public User Guide*:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
          "Effect": "Allow",
          "Action": [
             "ecr-public:GetAuthorizationToken",
             "sts:GetServiceBearerToken"
          ],
          "Resource": "*"
       }
    ]
}
```

Once these permissions are attached to your instance profile, Elastic Beanstalk will automatically authenticate with Amazon ECR Public registries. You can reference Amazon ECR Public images using the standard `public.ecr.aws/registry-alias/repository-name:tag` format in your `Dockerrun.aws.json` file or Dockerfile.

Configuring Elastic Beanstalk Docker environments

This chapter explains additional configuration information for all of the supported Docker platform branches, including the ECS managed Docker platform branch. Unless a specific platform branch or platform branch component is identified in a section, it applies to all environments that are running supported Docker and ECS manged Docker platforms.

Note

If your Elastic Beanstalk environment uses an Amazon Linux AMI Docker platform version (preceding Amazon Linux 2), be sure to read the additional information in the section called "Docker configuration on Amazon Linux AMI (preceding Amazon Linux 2)".

Sections

Configuring software in Docker environments

You can use the Elastic Beanstalk console to configure the software running on your environment's instances.

To configure your Docker environment in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. Make necessary configuration changes.
6. To save the changes choose **Apply** at the bottom of the page.

For information about configuring software settings in any environment, see the section called "Environment variables and software settings". The following sections cover Docker specific information.

Container options

The **Container options** section has platform-specific options. For Docker environments, it lets you choose whether or not your environment includes the NGINX proxy server.

Environments with Docker Compose

If you manage your Docker environment with Docker Compose, Elastic Beanstalk assumes that you run a proxy server as a container. Therefore it defaults to **None** for the **Proxy server** setting, and Elastic Beanstalk does not provide an NGINX configuration.

Note

Even if you select **NGINX** as a proxy server, this setting is ignored in an environment with Docker Compose. The **Proxy server** setting still defaults to **None**.

Since the NGINX web server proxy is disabled for the Docker on Amazon Linux 2 platform with Docker Compose, you must follow the instructions for generating logs for enhanced health reporting. For more information, see Generating logs for enhanced health reporting with Docker Compose.

Environment properties (environment variables)

You can use environment properties, (also known as environment variables), to pass values, such endpoints, debug settings, and other information to your application. The **Environment variables** section of the console lets you specify environment variables on the EC2 instances that are running your application. Environment variables are passed in as key-value pairs to the application.

Your application code running in a container can refer to an environment variable by name and read its value. The source code that reads these environment variables will vary by programming language. You can find instructions for reading environment variable values in the programming languages that Elastic Beanstalk managed platforms support in the respective platform topic. For a list of links to these topics, see the section called "Environment variables and software settings".

Secrets and parameters in Elastic Beanstalk environment variables

Elastic Beanstalk offers the ability to reference AWS Secrets Manager and AWS Systems Manager Parameter Store data in environment variables. This is a secure option for your application to natively access secrets and parameters stored by these services without having to manage API calls to them. Your Elastic Beanstalk Docker and ECS managed Docker platforms must be a version released on or after March 26, 2025 to support this feature. For more information about using environment variables to reference secrets, see Fetching secrets and parameters to Elastic Beanstalk environment variables.

Environments with Docker Compose

If you manage your Docker environment with Docker Compose, you must make some additional configuration to retrieve the environment variables in the containers. In order for the executables running in your container to access these environment variables, you must reference them in the `docker-compose.yml`. For more information see Referencing environment variables in containers.

Referencing environment variables in containers

If you are using the Docker Compose tool on the Amazon Linux 2 Docker platform, Elastic Beanstalk generates a Docker Compose environment file called `.env` in the root directory of your application project. This file stores the environment variables you configured for Elastic Beanstalk.

Note

If you include a `.env` file in your application bundle, Elastic Beanstalk will not generate an `.env` file.

In order for a container to reference the environment variables you define in Elastic Beanstalk, you must follow one or both of these configuration approaches.

- Add the `.env` file generated by Elastic Beanstalk to the `env_file` configuration option in the `docker-compose.yml` file.
- Directly define the environment variables in the `docker-compose.yml` file.

The following files provide an example. The sample `docker-compose.yml` file demonstrates both approaches.

- If you define environment properties `DEBUG_LEVEL=1` and `LOG_LEVEL=error`, Elastic Beanstalk generates the following `.env` file for you:

```
DEBUG_LEVEL=1
LOG_LEVEL=error
```

- In this `docker-compose.yml` file, the `env_file` configuration option points to the `.env` file, and it also defines the environment variable `DEBUG=1` directly in the `docker-compose.yml` file.

```
services:
  web:
    build: .
    environment:
      - DEBUG=1
    env_file:
      - .env
```

Notes

- If you set the same environment variable in both files, the variable defined in the `docker-compose.yml` file has higher precedence than the variable defined in the `.env` file.
- Be careful to not leave spaces between the equal sign (=) and the value assigned to your variable in order to prevent spaces from being added to the string.

To learn more about environment variables in Docker Compose, see Environment variables in Compose

Using interpolate feature for environment variables with Docker Compose

Starting with the July 28, 2023 platform release, the *Docker Amazon Linux 2* platform branch offers the Docker Compose *interpolation* feature. With this feature, values in a Compose file can be set by variables and interpolated at runtime. For more information about this feature, see Interpolation on the Docker documentation website.

Important

If you'd like to use this feature with your applications, be aware that you'll need to implement an approach that uses platform hooks.
This is necessary due a mitigation that we implemented in the platform engine. This mitigation ensures backward compatibility for customers that aren't aware of the new interpolation feature and have existing applications that use environment variables with the $ character. The updated platform engine escapes the interpolation by default by replacing the $ character with $$ characters.

The following is an example of a platform hook script that you can set up to allow use of the interpolation feature.

```
#!/bin/bash

: '
example data format in .env file
```

```
key1=value1
key2=value2
'
envfile="/var/app/staging/.env"
tempfile=$(mktemp)

while IFS= read -r line; do
  # split each env var string at '='
  split_str=(${line//=/ })
  if [ ${#split_str[@]} -eq 2 ]; then
    # replace '$$' with '$'
    replaced_str=${split_str[1]//\$\$/\$}
    # update the value of env var using ${replaced_str}
    line="${split_str[0]}=${replaced_str}"
  fi
  # append the updated env var to the tempfile
  echo "${line}" #"${tempfile}"
done < "${envfile}"
# replace the original .env file with the tempfile
mv "${tempfile}" "${envfile}"
```

Place the platform hooks under both of these directories:

- `.platform/confighooks/predeploy/`
- `.platform/hooks/predeploy/`

For more information, see Platform hooks in the *Extending Linux platforms* topic of this guide.

Generating logs for enhanced health reporting with Docker Compose

The Elastic Beanstalk health agent provides operating system and application health metrics for Elastic Beanstalk environments. It relies on web server log formats that relay information in a specific format.

Elastic Beanstalk assumes that you run a web server proxy as a container. As a result the NGINX web server proxy is disabled for Docker environments running Docker Compose. You must configure your server to write logs in the location and format that the Elastic Beanstalk health agent uses. Doing so allows you to make full use of enhanced health reporting, even if the web server proxy is disabled.

For instructions on how to do this, see Web server log configuration

Docker container customized logging with Docker Compose

In order to efficiently troubleshoot issues and monitor your containerized services, you can request instance logs from Elastic Beanstalk through the environment management console or the EB CLI. Instance logs are comprised of bundle logs and tail logs, combined and packaged to allow you to view logs and recent events in an efficient and straightforward manner.

Elastic Beanstalk creates log directories on the container instance, one for each service defined in the `docker-compose.yml` file, at `/var/log/eb-docker/containers/<service name>`. If you are using the Docker Compose feature on the Amazon Linux 2 Docker platform, you can mount these directories to the location within the container file structure where logs are written. When you mount log directories for writing log data, Elastic Beanstalk can gather log data from these directories.

To configure your service's logs files to be retrievable tail files and bundle logs

1. Edit the `docker-compose.yml` file.
2. Under the `volumes` key for your service add a bind mount to be the following:

   `"${EB_LOG_BASE_DIR}/<service name>:<log directory inside container>`

   In the sample `docker-compose.yml` file below:

   - `nginx-proxy` is `<service name>`
   - `/var/log/nginx` is `<log directory inside container>`

   ```
   services:
     nginx-proxy:
       image: "nginx"
       volumes:
         - "${EB_LOG_BASE_DIR}/nginx-proxy:/var/log/nginx"
   ```

- The `var/log/nginx` directory contains the logs for the *nginx-proxy* service in the container, and it will be mapped to the `/var/log/eb-docker/containers/nginx-proxy` directory on the host.
- All of the logs in this directory are now retrievable as bundle and tail logs through Elastic Beanstalk's request instance logs functionality.

Notes

- *${EB_LOG_BASE_DIR}* is an environment variable set by Elastic Beanstalk with the value `/var/log/eb-docker/containers`.
- Elastic Beanstalk automatically creates the `/var/log/eb-docker/containers/<service name>` directory for each service in the `docker-compose.yml`file.

Docker images

The Docker and ECS managed Docker platform branches for Elastic Beanstalk support the use of Docker images stored in a public or private online image repository.

Specify images by name in `Dockerrun.aws.json`. Note these conventions:

- Images in official repositories on Docker Hub use a single name (for example, `ubuntu` or `mongo`).
- Images in other repositories on Docker Hub are qualified with an organization name (for example, `amazon/amazon-ecs-agent`).
- Images in other online repositories are qualified further by a domain name (for example, `quay.io/assemblyline/ubuntu` or `account-id.dkr.ecr.us-east-2.amazonaws.com/ubuntu:trusty`).

For environments using the Docker platform only, you can also build your own image during environment creation with a Dockerfile. See Building custom images with a Dockerfile for details. The ECS managed Docker platform doesn't support this functionality.

Configuring managed updates for Docker environments

With managed platform updates, you can configure your environment to automatically update to the latest version of a platform on a schedule.

In the case of Docker environments, you might want to decide if an automatic platform update should happen across Docker versions—when the new platform version includes a new Docker version. Elastic Beanstalk supports managed platform updates across Docker versions when updating from an environment running a Docker platform version newer than 2.9.0. When a new platform version includes a new version of Docker, Elastic Beanstalk increments the minor update version number. Therefore, to allow managed platform updates across Docker versions, enable managed platform updates for both minor and patch version updates. To prevent managed platform updates across Docker versions, enable managed platform updates to apply patch version updates only.

For example, the following configuration file enables managed platform updates at 9:00 AM UTC each Tuesday for both minor and patch version updates, thereby allowing for managed updates across Docker versions:

Example .ebextensions/managed-platform-update.config

```
option_settings:
  aws:elasticbeanstalk:managedactions:
    ManagedActionsEnabled: true
    PreferredStartTime: "Tue:09:00"
  aws:elasticbeanstalk:managedactions:platformupdate:
    UpdateLevel: minor
```

For environments running Docker platform versions 2.9.0 or earlier, Elastic Beanstalk never performs managed platform updates if the new platform version includes a new Docker version.

Docker configuration namespaces

You can use a configuration file to set configuration options and perform other instance configuration tasks during deployments. Configuration options can be platform specific or apply to all platforms in the Elastic Beanstalk service as a whole. Configuration options are organized into *namespaces*.

Note

This information only applies to Docker environment that are not running Docker Compose. This option has a different behavior with Docker environments that run Docker Compose. For further information on proxy services with Docker Compose see Container options.

The Docker platform supports options in the following namespaces, in addition to the options supported for all Elastic Beanstalk environments:

- `aws:elasticbeanstalk:environment:proxy` – Choose the proxy server for your environment. Docker supports either running Nginx or no proxy server.

The following example configuration file configures a Docker environment to run no proxy server.

Example .ebextensions/docker-settings.config

```
option_settings:
  aws:elasticbeanstalk:environment:proxy:
    ProxyServer: none
```

Docker configuration on Amazon Linux AMI (preceding Amazon Linux 2)

If your Elastic Beanstalk Docker environment uses an Amazon Linux AMI platform version (preceding Amazon Linux 2), read the additional information in this section.

Using an authentication file for a private repository

This information is relevant to you if you are using images from a private repository. Beginning with Docker version 1.7, the **docker login** command changed the name of the authentication file, and the format of the file. Amazon Linux AMI Docker platform versions (preceding Amazon Linux 2) require the older `~/.dockercfg` format configuration file.

With Docker version 1.7 and later, the **docker login** command creates the authentication file in `~/.docker/config.json` in the following format.

```
{
    "auths":{
      "server":{
        "auth":"key"
      }
    }
  }
```

With Docker version 1.6.2 and earlier, the **docker login** command creates the authentication file in `~/.dockercfg` in the following format.

```
{
    "server" :
    {
      "auth" : "auth_token",
      "email" : "email"
    }
  }
```

To convert a `config.json` file, remove the outer `auths` key, add an `email` key, and flatten the JSON document to match the old format.

On Amazon Linux 2 Docker platform versions, Elastic Beanstalk uses the newer authentication file name and format. If you're using an Amazon Linux 2 Docker platform version, you can use the authentication file that the **docker login** command creates without any conversion.

Configuring additional storage volumes

For improved performance on Amazon Linux AMI, Elastic Beanstalk configures two Amazon EBS storage volumes for your Docker environment's Amazon EC2 instances. In addition to the root volume provisioned for all Elastic Beanstalk environments, a second 12GB volume named `xvdcz` is provisioned for image storage on Docker environments.

If you need more storage space or increased IOPS for Docker images, you can customize the image storage volume by using the `BlockDeviceMapping` configuration option in the aws:autoscaling:launchconfiguration namespace.

For example, the following configuration file increases the storage volume's size to 100 GB with 500 provisioned IOPS:

Example .ebextensions/blockdevice-xvdcz.config

```
option_settings:
  aws:autoscaling:launchconfiguration:
    BlockDeviceMappings: /dev/xvdcz=:100::io1:500
```

If you use the `BlockDeviceMappings` option to configure additional volumes for your application, you should include a mapping for `xvdcz` to ensure that it is created. The following example configures two volumes, the image storage volume `xvdcz` with default settings and an additional 24 GB application volume named `sdh`:

Example .ebextensions/blockdevice-sdh.config

```
option_settings:
  aws:autoscaling:launchconfiguration:
    BlockDeviceMappings: /dev/xvdcz=:12:true:gp2,/dev/sdh=:24
```

Note

When you change settings in this namespace, Elastic Beanstalk replaces all instances in your environment with instances running the new configuration. See Configuration changes for details.

Legacy platforms

This chapter lists content related to previous Docker platforms that are no longer supported by AWS Elastic Beanstalk. The topics listed here remain in this document as a reference for any customers that used these features or components prior to their retirement.

Topics

Migrating to Elastic Beanstalk Docker running on Amazon Linux 2 from Multi-container Docker running on Amazon Linux

Prior to the release of the *ECS Running on 64bit Amazon Linux 2* platform branch, Elastic Beanstalk offered an alternate migration path to Amazon Linux 2 for customers with environments based on the *Multi-container Docker running on 64bit Amazon Linux* platform branch. This topic describes that migration path, and remains in this document as a reference for any customers that completed that migration path.

We now recommend that customers with environments based on the *Multi-container Docker running on 64bit Amazon Linux* platform branch migrate to the *ECS Running on 64bit Amazon Linux 2* platform branch. Unlike the alternate migration path, this approach continues to use Amazon ECS to coordinate container deployments to ECS managed Docker environments. This aspect allows a more straightforward approach. No changes to the source code are required, and the same `Dockerrun.aws.json` v2 is supported. For more information, see Migrating your Elastic Beanstalk application from ECS managed Multi-container Docker on AL1 to ECS on Amazon Linux 2023.

Legacy Migration from Multi-container Docker on Amazon Linux to the Docker Amazon Linux 2 platform branch

You can migrate your applications running on the Multi-container Docker platform on Amazon Linux AMI to the Amazon Linux 2 Docker platform. The Multi-container Docker platform on Amazon Linux AMI requires that you specify prebuilt application images to run as containers. After migrating, you will no longer have this limitation, because the Amazon Linux 2 Docker platform also allows Elastic Beanstalk to build your container images during deployment. Your applications will continue to run in multi-container environments with the added benefits from the Docker Compose tool.

Docker Compose is tool for defining and running multi-container Docker applications. To learn more about Docker Compose and how to install it, see the Docker sites Overview of Docker Compose and Install Docker Compose.

The `docker-compose.yml` file

The Docker Compose tool uses the `docker-compose.yml` file for configuration of your application services. This file replaces your `Dockerrun.aws.json v2` file in your application project directory and application source bundle. You create the `docker-compose.yml` file manually, and will find it helpful to reference your `Dockerrun.aws.json v2` file for most of the parameter values.

Below is an example of a `docker-compose.yml` file and the corresponding `Dockerrun.aws.json v2` file for the same application. For more information on the `docker-compose.yml` file, see Compose file reference. For more information on the `Dockerrun.aws.json v2` file, see `Dockerrun.aws.json` v2.

`docker-compose.yml`	`Dockerrun.aws.json v2`

`docker-compose.yml`

```
version: '2.4'
services:
  php-app:
    image: "php:fpm"
    volumes:
      - "./php-app:/var/www/html:ro
"
```

`Dockerrun.aws.json v2`

```
{
  "AWSEBDockerrunVersion": 2,
  "volumes": [
    {
      "name": "php-app",
      "host": {
```

docker-compose.yml	Dockerrun.aws.json v2

```
      - "${EB_LOG_BASE_DIR}/php-app
:/var/log/sample-app"
    mem_limit: 128m
    environment:
      Container: PHP
  nginx-proxy:
    image: "nginx"
    ports:
      - "80:80"
    volumes:
      - "./php-app:/var/www/html:ro
"
      - "./proxy/conf.d:/etc/nginx/
conf.d:ro"
      - "${EB_LOG_BASE_DIR}/nginx-p
roxy:/var/log/nginx"
    mem_limit: 128m
    links:
      - php-app
```
	```
	        "sourcePath": "/var/app/
	current/php-app"
	      }
	    },
	    {
	      "name": "nginx-proxy-conf",
	      "host": {
	        "sourcePath": "/var/app/
	current/proxy/conf.d"
	      }
	    }
	  ],
	  "containerDefinitions": [
	    {
	      "name": "php-app",
	      "image": "php:fpm",
	      "environment": [
	        {
	          "name": "Container",
	          "value": "PHP"
	        }
	      ],
	      "essential": true,
	      "memory": 128,
	      "mountPoints": [
	        {
	          "sourceVolume": "php-app"
	,
	          "containerPath": "/var/www
	/html",
	          "readOnly": true
	        }
	      ]
	    },
	    {
	      "name": "nginx-proxy",
	      "image": "nginx",
	      "essential": true,
	      "memory": 128,
	      "portMappings": [
	        {
	          "hostPort": 80,
	```

docker-compose.yml	Dockerrun.aws.json v2
	(continued below)

```
          "containerPort": 80
        }
      ],
      "links": [
        "php-app"
      ],
      "mountPoints": [
        {
          "sourceVolume": "php-app"
,
          "containerPath": "/var/www
/html",
          "readOnly": true
        },
        {
          "sourceVolume": "nginx-pr
oxy-conf",
          "containerPath": "/etc/ngi
nx/conf.d",
          "readOnly": true
        },
        {
          "sourceVolume": "awseb-lo
gs-nginx-proxy",
          "containerPath": "/var/log
/nginx"
        }
      ]
    }
  ]
}
```

Additional Migration Considerations

The Docker Amazon Linux 2 platform and Multi-container Docker Amazon Linux AMI platform implement environment properties differently. These two platforms also have different log directories that Elastic Beanstalk creates for each of their containers. After you migrate from the

Amazon Linux AMI Multi-container Docker platform, you will need to be aware of these different implementations for your new Amazon Linux 2 Docker platform environment.

Area	Docker platform on Amazon Linux 2 with Docker Compose	Multi-container Docker platform on Amazon Linux AMI
Environment properties	In order for your containers to access environment properties you must add a reference to the `.env` file in the `docker-compose.yml` file. Elastic Beanstalk generates the `.env` file, listing each of the properties as environment variables. For more information see Referencing environment variables in containers.	Elastic Beanstalk can directly pass environment properties to the container. Your code running in the container can access these properties as environment variables without any additional configuration.
Log directories	For each container Elastic Beanstalk creates a log directory called `/var/log/eb-docker/containers/ *<service name>*` (or `${EB_LOG_BASE_DIR}/<service name>`). For more information see Docker container customized logging with Docker Compose.	For each container, Elastic Beanstalk creates a log directory called `/var/log/containers/ *<containername>*` . For more information see `mountPoints` field in Container definition format.

Migration Steps

To migrate to the Amazon Linux 2 Docker platform

1. Create the `docker-compose.yml` file for your application, based on its existing `Dockerrun.aws.json v2` file. For more information see the above section The `docker-compose.yml` file.
2. In your application project folder's root directory, replace the `Dockerrun.aws.json v2` file with the `docker-compose.yml` you just created.

   Your directory structure should be as follows.

```
~/myApplication
|-- docker-compose.yml
|-- .ebextensions
|-- php-app
|-- proxy
```

3. Use the **eb init** command to configure your local directory for deployment to Elastic Beanstalk.

```
~/myApplication$ eb init -p docker application-name
```

4. Use the **eb create** command to create an environment and deploy your Docker image.

```
~/myApplication$ eb create environment-name
```

5. If your app is a web application, after your environment launches, use the **eb open** command to view it in a web browser.

```
~/myApplication$ eb open environment-name
```

6. You can display the status of your newly created environment using the **eb status** command.

```
~/myApplication$ eb status environment-name
```

Preconfigured Docker GlassFish containers on Elastic Beanstalk

Note

On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information about migrating to a current and fully supported Amazon Linux 2023 platform branch, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

The Preconfigured Docker GlassFish platform branch that runs on the Amazon Linux AMI (AL1) is no longer supported. To migrate your GlassFish application to a supported Amazon Linux 2023 platform, deploy GlassFish and your application code to an Amazon Linux 2023 Docker image. For more information, see the following topic, the section called "Tutorial - GlassFish on Docker: path to Amazon Linux 2023".

Getting started with preconfigured Docker containers - on Amazon Linux AMI (preceding Amazon Linux 2)

This section shows you how to develop an example application locally and then deploy your application to Elastic Beanstalk with a preconfigured Docker container.

Set up your local development environment

For this walk-through we use a GlassFish example application.

To set up your environment

1. Create a new folder for the example application.

```
~$ mkdir eb-preconf-example
~$ cd eb-preconf-example
```

2. Download the example application code into the new folder.

```
~$ wget https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/samples/docker-glassfish-v1.zip
~$ unzip docker-glassfish-v1.zip
~$ rm docker-glassfish-v1.zip
```

Develop and test locally

To develop an example GlassFish application

1. Add a `Dockerfile` to your application's root folder. In the file, specify the AWS Elastic Beanstalk Docker base image to be used to run your local preconfigured Docker container. You'll later deploy your application to an Elastic Beanstalk Preconfigured Docker GlassFish platform version. Choose the Docker base image that this platform version uses. To find out the current Docker image of the platform version, see the Preconfigured Docker section of the *AWS Elastic Beanstalk Supported Platforms* page in the *AWS Elastic Beanstalk Platforms* guide.

Example ~/Eb-preconf-example/Dockerfile

```
# For Glassfish 5.0 Java 8
FROM amazon/aws-eb-glassfish:5.0-al-onbuild-2.11.1
```

For more information about using a `Dockerfile`, see Preparing your Docker image for deployment to Elastic Beanstalk.

2. Build the Docker image.

```
~/eb-preconf-example$ docker build -t my-app-image .
```

3. Run the Docker container from the image.

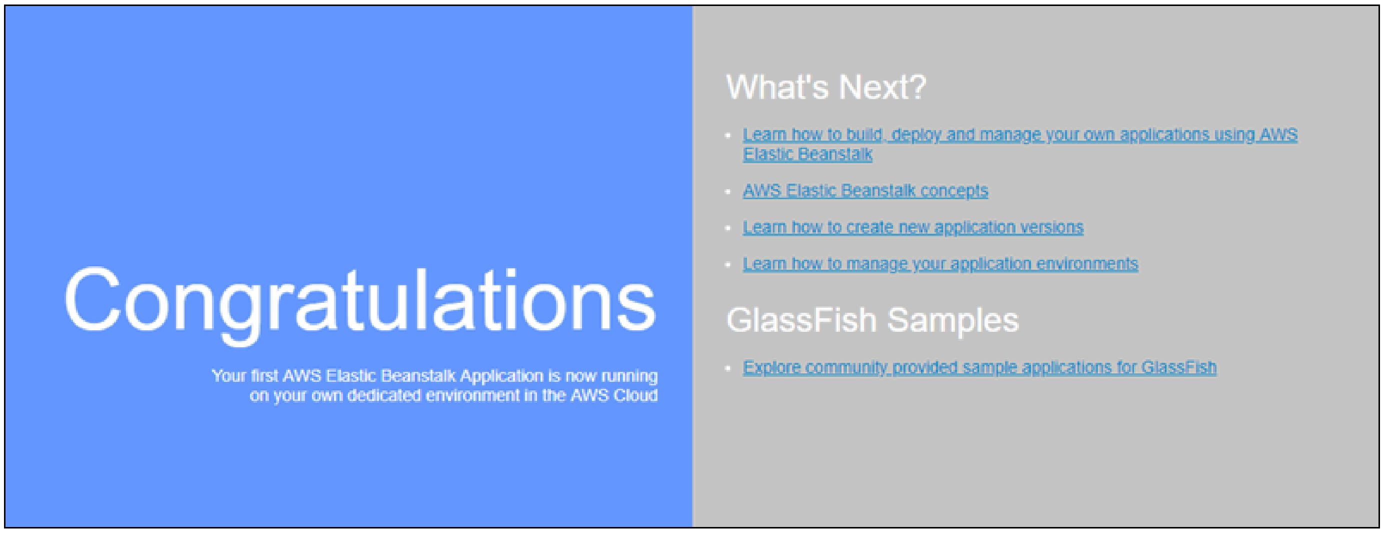
Note

You must include the `-p` flag to map port 8080 on the container to the localhost port 3000. Elastic Beanstalk Docker containers always expose the application on port 8080 on the container. The `-it` flags run the image as an interactive process. The `--rm` flag cleans up the container file system when the container exits. You can optionally include the `-d` flag to run the image as a daemon.

```
$ docker run -it --rm -p 3000:8080 my-app-image
```

4. To view the example application, type the following URL into your web browser.

```
http://localhost:3000
```



Congratulations

Your first AWS Elastic Beanstalk Application is now running on your own dedicated environment in the AWS Cloud

What's Next?

- Learn how to build, deploy and manage your own applications using AWS Elastic Beanstalk
- AWS Elastic Beanstalk concepts
- Learn how to create new application versions
- Learn how to manage your application environments

GlassFish Samples

- Explore community provided sample applications for GlassFish

Deploy to Elastic Beanstalk

After testing your application, you are ready to deploy it to Elastic Beanstalk.

To deploy your application to Elastic Beanstalk

1. In your application's root folder, rename the `Dockerfile` to `Dockerfile.local`. This step is required for Elastic Beanstalk to use the `Dockerfile` that contains the correct instructions for Elastic Beanstalk to build a customized Docker image on each Amazon EC2 instance in your Elastic Beanstalk environment.

   Note

   You do not need to perform this step if your `Dockerfile` includes instructions that modify the platform version's base Docker image. You do not need to use a `Dockerfile` at all if your `Dockerfile` includes only a `FROM` line to specify the base image from which to build the container. In that situation, the `Dockerfile` is redundant.

2. Create an application source bundle.

```
~/eb-preconf-example$ zip myapp.zip -r *
```

3. Open the Elastic Beanstalk console with this preconfigured link: console.aws.amazon.com/elasticbeanstalk/home#/newApplication?applicationName=tutorials&environmentType=LoadBalanced
4. For **Platform**, under **Preconfigured – Docker**, choose **Glassfish**.
5. For **Application code**, choose **Upload your code**, and then choose **Upload**.
6. Choose **Local file**, choose **Browse**, and then open the application source bundle you just created.
7. Choose **Upload**.
8. Choose **Review and launch**.
9. Review the available settings, and then choose **Create app**.
10. When the environment is created, you can view the deployed application. Choose the environment URL that is displayed at the top of the console dashboard.

Deploying a GlassFish application to the Docker platform: a migration path to Amazon Linux 2023

The goal of this tutorial is to provide customers using the Preconfigured Docker GlassFish platform (based on Amazon Linux AMI) with a migration path to Amazon Linux 2023. You can migrate your GlassFish application to Amazon Linux 2023 by deploying GlassFish and your application code to an Amazon Linux 2023 Docker image.

The tutorial walks you through using the AWS Elastic Beanstalk Docker platform to deploy an application based on the Java EE GlassFish application server to an Elastic Beanstalk environment.

We demonstrate two approaches to building a Docker image:

- **Simple** – Provide your GlassFish application source code and let Elastic Beanstalk build and run a Docker image as part of provisioning your environment. This is easy to set up, at a cost of increased instance provisioning time.
- **Advanced** – Build a custom Docker image containing your application code and dependencies, and provide it to Elastic Beanstalk to use in your environment. This approach is slightly more involved, and decreases the provisioning time of instances in your environment.

Prerequisites

This tutorial assumes that you have some knowledge of basic Elastic Beanstalk operations, the Elastic Beanstalk command line interface (EB CLI), and Docker. If you haven't already, follow the instructions in Learn how to get started with Elastic Beanstalk to launch your first Elastic Beanstalk environment. This tutorial uses the EB CLI, but you can also create environments and upload applications by using the Elastic Beanstalk console.

To follow this tutorial, you will also need the following Docker components:

- A working local installation of Docker. For more information, see Get Docker on the Docker documentation website.
- Access to Docker Hub. You will need to create a Docker ID to access the Docker Hub. For more information, see Share the application on the Docker documentation website.

To learn more about configuring Docker environments on Elastic Beanstalk platforms, see Preparing your Docker image for deployment to Elastic Beanstalk in this same chapter.

Simple example: provide your application code

This is an easy way to deploy your GlassFish application. You provide your application source code together with the `Dockerfile` included in this tutorial. Elastic Beanstalk builds a Docker image that includes your application and the GlassFish software stack. Then Elastic Beanstalk runs the image on your environment instances.

An issue with this approach is that Elastic Beanstalk builds the Docker image locally whenever it creates an instance for your environment. The image build increases instance provisioning time. This impact isn't limited to initial environment creation—it happens during scale-out actions too.

To launch an environment with an example GlassFish application

1. Download the example `docker-glassfish-al2-v1.zip`, and then expand the `.zip` file into a directory in your development environment.

   ```
   ~$ curl https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/samples/docker-
   glassfish-al2-v1.zip --output docker-glassfish-al2-v1.zip
   ~$ mkdir glassfish-example
   ~$ cd glassfish-example
   ~/glassfish-example$ unzip ../docker-glassfish-al2-v1.zip
   ```

   Your directory structure should be as follows.

   ```
   ~/glassfish-example
   |-- Dockerfile
   |-- Dockerrun.aws.json
   |-- glassfish-start.sh
   |-- index.jsp
   |-- META-INF
   |   |-- LICENSE.txt
   |   |-- MANIFEST.MF
   |   `-- NOTICE.txt
   |-- robots.txt
   `-- WEB-INF
       `-- web.xml
   ```

   The following files are key to building and running a Docker container in your environment:

   - `Dockerfile` – Provides instructions that Docker uses to build an image with your application and required dependencies.

- `glassfish-start.sh` – A shell script that the Docker image runs to start your application.
- `Dockerrun.aws.json` – Provides a logging key, to include the GlassFish application server log in log file requests. If you aren't interested in GlassFish logs, you can omit this file.

2. Configure your local directory for deployment to Elastic Beanstalk.

```
~/glassfish-example$ eb init -p docker glassfish-example
```

3. (Optional) Use the **eb local run** command to build and run your container locally.

```
~/glassfish-example$ eb local run --port 8080
```

> **Note**
>
> To learn more about the **eb local** command, see the section called "**eb local**". The command isn't supported on Windows. Alternatively, you can build and run your container with the **docker build** and **docker run** commands. For more information, see the Docker documentation.

4. (Optional) While your container is running, use the **eb local open** command to view your application in a web browser. Alternatively, open http://localhost:8080/ in a web browser.

```
~/glassfish-example$ eb local open
```

5. Use the **eb create** command to create an environment and deploy your application.

```
~/glassfish-example$ eb create glassfish-example-env
```

6. After your environment launches, use the **eb open** command to view it in a web browser.

```
~/glassfish-example$ eb open
```

When you're done working with the example, terminate the environment and delete related resources.

```
~/glassfish-example$ eb terminate --all
```

Advanced example: provide a prebuilt Docker image

This is a more advanced way to deploy your GlassFish application. Building on the first example, you create a Docker image containing your application code and the GlassFish software stack, and push it to Docker Hub. After you've done this one-time step, you can launch Elastic Beanstalk environments based on your custom image.

When you launch an environment and provide your Docker image, instances in your environment download and use this image directly and don't need to build a Docker image. Therefore, instance provisioning time is decreased.

Notes

- The following steps create a publicly available Docker image.
- You will use Docker commands from your local Docker installation, along with your Docker Hub credentials. For more information, see the preceding *Prerequisites* section in this topic.

To launch an environment with a prebuilt GlassFish application Docker image

1. Download and expand the example `docker-glassfish-al2-v1.zip` as in the previous simple example. If you've completed that example, you can use the directory you already have.
2. Build a Docker image and push it to Docker Hub. Enter your Docker ID for *`docker-id`* to sign in to Docker Hub.

   ```
   ~/glassfish-example$ docker build -t docker-id/beanstalk-glassfish-example:latest .
   ~/glassfish-example$ docker push docker-id/beanstalk-glassfish-example:latest
   ```

   Note

   Before pushing your image, you might need to run **docker login**. You will be prompted for your Docker Hub credentials if you run the command without parameters.

3. Create an additional directory.

   ```
   ~$ mkdir glassfish-prebuilt
   ~$ cd glassfish-prebuilt
   ```

4. Copy the following example into a file named `Dockerrun.aws.json`.

   Example ~/glassfish-prebuilt/Dockerrun.aws.json

   ```
   {
     "AWSEBDockerrunVersion": "1",
     "Image": {
       "Name": "docker-username/beanstalk-glassfish-example"
     },
     "Ports": [
       {
         "ContainerPort": 8080,
         "HostPort": 8080
       }
     ],
     "Logging": "/usr/local/glassfish5/glassfish/domains/domain1/logs"
   }
   ```

5. Configure your local directory for deployment to Elastic Beanstalk.

   ```
   ~/glassfish-prebuilt$ eb init -p docker glassfish-prebuilt$
   ```

6. (Optional) Use the **eb local run** command to run your container locally.

   ```
   ~/glassfish-prebuilt$ eb local run --port 8080
   ```

7. (Optional) While your container is running, use the **eb local open** command to view your application in a web browser. Alternatively, open http://localhost:8080/ in a web browser.

   ```
   ~/glassfish-prebuilt$ eb local open
   ```

8. Use the **eb create** command to create an environment and deploy your Docker image.

   ```
   ~/glassfish-prebuilt$ eb create glassfish-prebuilt-env
   ```

9. After your environment launches, use the **eb open** command to view it in a web browser.

   ```
   ~/glassfish-prebuilt$ eb open
   ```

When you're done working with the example, terminate the environment and delete related resources.

```
~/glassfish-prebuilt$ eb terminate --all
```

Monitoring environments in Elastic Beanstalk

With Elastic Beanstalk health monitoring, you can verify application availability and create alerts that activate when metrics exceed your thresholds. You can use Elastic Beanstalk health monitoring in both the console and command line to track your environment's status.

Topics

Monitoring environment health in the AWS management console

You can access operational information about your application from the Elastic Beanstalk console. The console displays your environment's status and application health at a glance. In the console's **Environments** page and in each application's page, the environments on the list are color-coded to indicate status.

To monitor an environment in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Monitoring**.

The Monitoring page shows you overall statistics about your environment, such as CPU utilization and average latency. In addition to the overall statistics, you can view monitoring graphs that show resource usage over time. You can click any of the graphs to view more detailed information.

Note

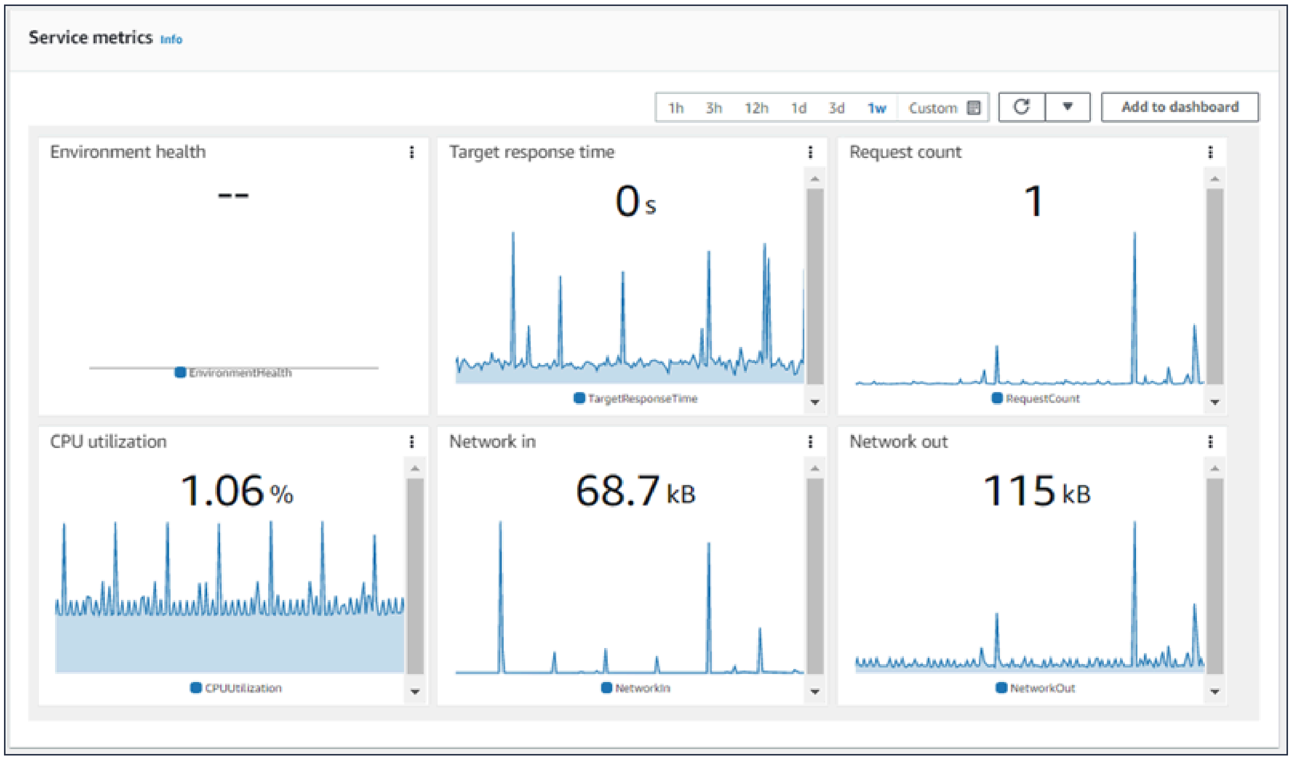
By default, only basic CloudWatch metrics are enabled, which return data in five-minute periods. You can enable more granular one-minute CloudWatch metrics by editing your environment's configuration settings.

Monitoring graphs

The **Monitoring** page shows an overview of health-related metrics for your environment. This includes the default set of metrics provided by Elastic Load Balancing and Amazon EC2, and graphs that show how the environment's health has changed over time.

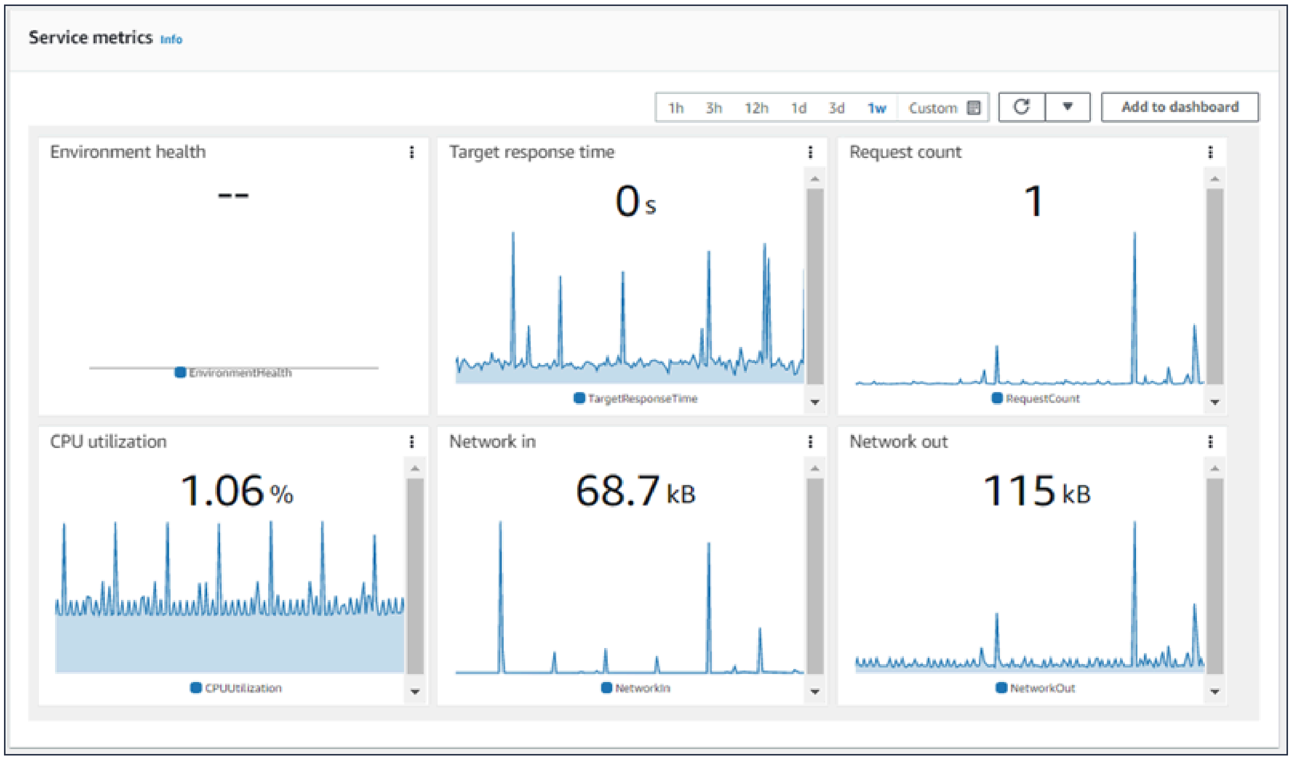
The bar above the graphs provides a variety of time intervals for you to select. For example, select **1w** to display information that spans over the last week. Or select **3h** to display information that spans over the last three hours.

For a greater variety of time interval selections, choose **Custom**. From here you have two range options: *Absolute* or *Relative*. The **Absolute** option allows you to specify a specific date range, such as *January 1, 2023 to June 30, 2023*. The **Relative** option allows to select an integer with a specific time unit: *Minutes*, *Hours, Days*, *Weeks*, or *Months*. Examples include *10 Hours*, *10 Days*, and *10 Months*.



Customizing the monitoring console

To create and view custom metrics you must use Amazon CloudWatch. With CloudWatch you can create custom dashboards to monitor your resources in a single view. Select **Add to dashboard** to navigate to the Amazon CloudWatch console from the **Monitoring** page. Amazon CloudWatch provides you the option to create a new dashboard or select an existing one. For more information, see Using Amazon CloudWatch dashboards in the *Amazon CloudWatch User Guide*.



Elastic Load Balancing and Amazon EC2 metrics are enabled for all environments.

With enhanced health, the EnvironmentHealth metric is enabled, and a graph is added to the monitoring console automatically. Enhanced health also adds the Health page to the management console. For a list of available enhanced health metrics, see Publishing Amazon CloudWatch custom metrics for an environment.

Using the EB CLI to monitor environment health

The Elastic Beanstalk Command Line Interface (EB CLI) is a command line tool for managing AWS Elastic Beanstalk environments. You also can use the EB CLI to monitor your environment's health in real time and with more granularity than is currently available in the Elastic Beanstalk console

After installing and configuring the EB CLI, you can launch a new environment and deploy your code to it with the **eb create** command. If you already have an environment that you created in the Elastic Beanstalk console, you can attach the EB CLI to it by running **eb init** in a project folder and following the prompts (the project folder can be empty).

Important

Ensure that you are using the latest version of the EB CLI by running `pip install` with the `--upgrade` option:

```
$ sudo pip install --upgrade awsebcli
```

For complete EB CLI installation instructions, see Install EB CLI with setup script (recommended).

To use the EB CLI to monitor your environment's health, you must first configure a local project folder by running **eb init** and following the prompts. For complete instructions, see Configure the EB CLI.

If you already have an environment running in Elastic Beanstalk and want to use the EB CLI to monitor its health, follow these steps to attach it to the existing environment.

To attach the EB CLI to an existing environment

1. Open a command line terminal and navigate to your user folder.
2. Create and open a new folder for your environment.
3. Run the **eb init** command, and then choose the application and environment whose health you want to monitor. If you have only one environment running the application you choose, the EB CLI will select it automatically and you won't need to choose the environment, as shown in the following example.

```
~/project$ eb init
Select an application to use
1) elastic-beanstalk-example
2) [ Create new Application ]
(default is 2): 1
Select the default environment.
You can change this later by typing "eb use [environment_name]".
1) elasticBeanstalkEx2-env
2) elasticBeanstalkExa-env
(default is 1): 1
```

To monitor health by using the EB CLI

1. Open a command line and navigate to your project folder.
2. Run the **eb health** command to display the health status of the instances in your environment. In this example, there are five instances running in a Linux environment.

```
~/project $ eb health
 elasticBeanstalkExa-env                                 Ok
 2015-07-08 23:13:20
WebServer
    Ruby 2.1 (Puma)
  total      ok    warning  degraded  severe    info   pending  unknown
    5        5        0        0        0        0        0        0

  instance-id   status     cause
                                                health
    Overall     Ok
  i-d581497d    Ok
  i-d481497c    Ok
  i-136e00c0    Ok
  i-126e00c1    Ok
  i-8b2cf575    Ok

  instance-id   r/sec    %2xx   %3xx   %4xx   %5xx      p99      p90      p75
 p50     p10                                    requests
    Overall     671.8   100.0    0.0    0.0    0.0    0.003    0.002    0.001
 0.001   0.000
  i-d581497d    143.0    1430      0      0      0    0.003    0.002    0.001
 0.001   0.000
  i-d481497c    128.8    1288      0      0      0    0.003    0.002    0.001
 0.001   0.000
  i-136e00c0    125.4    1254      0      0      0    0.004    0.002    0.001
 0.001   0.000
  i-126e00c1    133.4    1334      0      0      0    0.003    0.002    0.001
 0.001   0.000
  i-8b2cf575    141.2    1412      0      0      0    0.003    0.002    0.001
 0.001   0.000

  instance-id   type       az   running     load 1  load 5      user%  nice%
 system%  idle%   iowait%                            cpu
  i-d581497d    t2.micro   1a   12 mins        0.0    0.04        6.2    0.0
 1.0   92.5       0.1
```

```
  i-d481497c    t2.micro   1a   12 mins       0.01    0.09        5.9    0.0
 1.6   92.4       0.1
  i-136e00c0    t2.micro   1b   12 mins       0.15    0.07        5.5    0.0
 0.9   93.2       0.0
  i-126e00c1    t2.micro   1b   12 mins       0.17    0.14        5.7    0.0
 1.4   92.7       0.1
  i-8b2cf575    t2.micro   1c   1 hour        0.19    0.08        6.5    0.0
 1.2   92.1       0.1

  instance-id   status     id   version              ago
                                        deployments
  i-d581497d    Deployed   1    Sample Application   12 mins
  i-d481497c    Deployed   1    Sample Application   12 mins
  i-136e00c0    Deployed   1    Sample Application   12 mins
  i-126e00c1    Deployed   1    Sample Application   12 mins
  i-8b2cf575    Deployed   1    Sample Application   1 hour
```

In this example, there is a single instance running in a Windows environment.

```
~/project $ eb health
 WindowsSampleApp-env                                 Ok
      2018-05-22 17:33:19
WebServer                                                IIS 10.0 running on 64bit
 Windows Server 2016/2.2.0
  total      ok    warning  degraded  severe    info   pending  unknown
    1        1        0        0        0        0        0        0

  instance-id           status     cause
                                                 health
    Overall             Ok
  i-065716fba0e08a351   Ok

  instance-id           r/sec    %2xx   %3xx   %4xx   %5xx      p99      p90
 p75     p50     p10                         requests
    Overall              13.7   100.0    0.0    0.0    0.0    1.403    0.970
 0.710   0.413   0.079
  i-065716fba0e08a351     2.4   100.0    0.0    0.0    0.0    1.102*   0.865
 0.601   0.413   0.091

  instance-id           type       az   running     % user time    % privileged
 time  % idle time                                  cpu
  i-065716fba0e08a351   t2.large   1b   4 hours             0.2
 0.1         99.7
```

```
  instance-id           status     id   version              ago
                                        deployments
  i-065716fba0e08a351   Deployed   2    Sample Application   4 hours
```

Reading the output

The output displays the name of the environment, the environment's overall health, and the current date at the top of the screen.

```
elasticBeanstalkExa-env                                  Ok
 2015-07-08 23:13:20
```

The next three lines display the type of environment ("WebServer" in this case), the configuration (Ruby 2.1 with Puma), and a breakdown of how many instances are in each of the seven states.

```
WebServer
 Ruby 2.1 (Puma)
  total      ok    warning  degraded  severe    info   pending  unknown
    5         5       0        0        0        0        0        0
```

The rest of the output is split into four sections. The first displays the *status* and the *cause* of the status for the environment overall, and then for each instance. The following example shows two instances in the environment with a status of `Info` and a cause indicating that a deployment has started.

```
  instance-id    status     cause
                                          health
    Overall      Ok
  i-d581497d     Info       Performing application deployment (running for 3 seconds)
  i-d481497c     Info       Performing application deployment (running for 3 seconds)
  i-136e00c0     Ok
  i-126e00c1     Ok
  i-8b2cf575     Ok
```

For information about health statuses and colors, see Health colors and statuses.

The **requests** section displays information from the web server logs on each instance. In this example, each instance is taking requests normally and there are no errors.

```
  instance-id    r/sec    %2xx   %3xx   %4xx   %5xx      p99      p90      p75     p50
    p10                                 requests
    Overall      13.7     100.0    0.0    0.0    0.0    1.403    0.970    0.710   0.413
  0.079
  i-d581497d     2.4      100.0    0.0    0.0    0.0    1.102*   0.865    0.601   0.413
  0.091
  i-d481497c     2.7      100.0    0.0    0.0    0.0    0.842*   0.788    0.480   0.305
  0.062
  i-136e00c0     4.1      100.0    0.0    0.0    0.0    1.520*   1.088    0.883   0.524
  0.104
  i-126e00c1     2.2      100.0    0.0    0.0    0.0    1.334*   0.791    0.760   0.344
  0.197
  i-8b2cf575     2.3      100.0    0.0    0.0    0.0    1.162*   0.867    0.698   0.477
  0.076
```

The **cpu** section shows operating system metrics for each instance. The output differs by operating system. Here is the output for Linux environments.

```
  instance-id   type       az   running     load 1  load 5      user%  nice%  system%
 idle%   iowait%                             cpu
  i-d581497d    t2.micro   1a   12 mins        0.0    0.03        0.2    0.0      0.0
 99.7        0.1
  i-d481497c    t2.micro   1a   12 mins        0.0    0.03        0.3    0.0      0.0
 99.7        0.0
  i-136e00c0    t2.micro   1b   12 mins        0.0    0.04        0.1    0.0      0.0
 99.9        0.0
  i-126e00c1    t2.micro   1b   12 mins       0.01    0.04        0.2    0.0      0.0
 99.7        0.1
  i-8b2cf575    t2.micro   1c   1 hour         0.0    0.01        0.2    0.0      0.1
 99.6        0.1
```

Here is the output for Windows environments.

```
  instance-id           type       az   running     % user time    % privileged time  %
 idle time
  i-065716fba0e08a351   t2.large   1b   4 hours             0.2                 0.0
     99.8
```

For information about the server and operating system metrics shown, see Instance metrics.

The final section, **deployments**, shows the deployment status of each instance. If a rolling deployment fails, you can use the deployment ID, status, and version label shown to identify instances in your environment that are running the wrong version.

```
  instance-id   status     id   version              ago
                                    deployments
  i-d581497d    Deployed   1    Sample Application   12 mins
  i-d481497c    Deployed   1    Sample Application   12 mins
  i-136e00c0    Deployed   1    Sample Application   12 mins
  i-126e00c1    Deployed   1    Sample Application   12 mins
  i-8b2cf575    Deployed   1    Sample Application   1 hour
```

Interactive health view

The **eb health** command displays a snapshot of your environment's health. To refresh the displayed information every 10 seconds, use the `--refresh` option.

```
$ eb health --refresh
 elasticBeanstalkExa-env                             Ok
 2015-07-09 22:10:04 (1 secs)
WebServer
          Ruby 2.1 (Puma)
  total      ok    warning  degraded  severe    info   pending  unknown
    5        5        0        0        0        0        0        0

  instance-id   status     cause
                                         health
    Overall     Ok
  i-bb65c145    Ok         Application deployment completed 35 seconds ago and took 26
 seconds
  i-ba65c144    Ok         Application deployment completed 17 seconds ago and took 25
 seconds
  i-f6a2d525    Ok         Application deployment completed 53 seconds ago and took 26
 seconds
  i-e8a2d53b    Ok         Application deployment completed 32 seconds ago and took 31
 seconds
  i-e81cca40    Ok

  instance-id   r/sec    %2xx   %3xx   %4xx   %5xx      p99      p90      p75     p50
   p10                                 requests
    Overall     671.8   100.0    0.0    0.0    0.0    0.003    0.002    0.001   0.001
 0.000
  i-bb65c145    143.0    1430      0      0      0    0.003    0.002    0.001   0.001
 0.000
  i-ba65c144    128.8    1288      0      0      0    0.003    0.002    0.001   0.001
 0.000
  i-f6a2d525    125.4    1254      0      0      0    0.004    0.002    0.001   0.001
 0.000
  i-e8a2d53b    133.4    1334      0      0      0    0.003    0.002    0.001   0.001
 0.000
  i-e81cca40    141.2    1412      0      0      0    0.003    0.002    0.001   0.001
 0.000

  instance-id   type        az   running     load 1  load 5      user%  nice%  system%
 idle%   iowait%                              cpu
  i-bb65c145    t2.micro   1a   12 mins        0.0    0.03        0.2    0.0      0.0
 99.7       0.1
  i-ba65c144    t2.micro   1a   12 mins        0.0    0.03        0.3    0.0      0.0
 99.7       0.0
  i-f6a2d525    t2.micro   1b   12 mins        0.0    0.04        0.1    0.0      0.0
 99.9       0.0
  i-e8a2d53b    t2.micro   1b   12 mins       0.01    0.04        0.2    0.0      0.0
 99.7       0.1
  i-e81cca40    t2.micro   1c   1 hour         0.0    0.01        0.2    0.0      0.1
 99.6       0.1

  instance-id   status     id   version              ago
                                    deployments
  i-bb65c145    Deployed   1    Sample Application   12 mins
  i-ba65c144    Deployed   1    Sample Application   12 mins
  i-f6a2d525    Deployed   1    Sample Application   12 mins
  i-e8a2d53b    Deployed   1    Sample Application   12 mins
  i-e81cca40    Deployed   1    Sample Application   1 hour

 (Commands: Help,Quit, # # # #)
```

This example shows an environment that has recently been scaled up from one to five instances. The scaling operation succeeded, and all instances are now passing health checks and are ready to take requests. In interactive mode, the health status updates every 10 seconds. In the upper-right corner, a timer ticks down to the next update.

In the lower-left corner, the report displays a list of options. To exit interactive mode, press **Q**. To scroll, press the arrow keys. To see a list of additional commands, press **H**.

Interactive health view options

When viewing environment health interactively, you can use keyboard keys to adjust the view and tell Elastic Beanstalk to replace or reboot individual instances. To see a list of available commands while viewing the health report in interactive mode, press **H** .

```
  up,down,home,end   Scroll vertically
  left,right         Scroll horizontally
  F                  Freeze/unfreeze data
  X                  Replace instance
  B                  Reboot instance
  <,>                Move sort column left/right
  -,+                Sort order descending/ascending
  P                  Save health snapshot data file
  Z                  Toggle color/mono mode
  Q                  Quit this program

  Views
  1                  All tables/split view
  2                  Status Table
  3                  Request Summary Table
  4                  CPU%/Load Table
  H                  This help menu


(press Q or ESC to return)
```

Basic health reporting

This topic explains the functionality offered by Elastic Beanstalk basic health.

AWS Elastic Beanstalk uses information from multiple sources to determine if your environment is available and processing requests from the Internet. An environment's health is represented by one of four colors, and is displayed on the environment overview page of the Elastic Beanstalk console. It's also available from the DescribeEnvironments API and by calling **eb status** with the EB CLI.

The basic health reporting system provides information about the health of instances in an Elastic Beanstalk environment based on health checks performed by Elastic Load Balancing for load-balanced environments, or Amazon Elastic Compute Cloud for single-instance environments.

In addition to checking the health of your EC2 instances, Elastic Beanstalk also monitors the other resources in your environment and reports missing or incorrectly configured resources that can cause your environment to become unavailable to users.

Metrics gathered by the resources in your environment is published to Amazon CloudWatch in five minute intervals. This includes operating system metrics from EC2, request metrics from Elastic Load Balancing. You can view graphs based on these CloudWatch metrics on the Monitoring page of the environment console. For basic health, these metrics are not used to determine an environment's health.

Topics

Health colors

Elastic Beanstalk reports the health of a web server environment depending on how the application running in it responds to the health check. Elastic Beanstalk uses one of four colors to describe status, as shown in the following table:

Color	Description
Grey	Your environment is being updated.
Green	Your environment has passed the most recent health check. At least one instance in your environment is available and taking requests.
Yellow	Your environment has failed one or more health checks. Some requests to your environment are failing.
Red	Your environment has failed three or more health checks, or an environment resource has become unavailable. Requests are consisten tly failing.

These descriptions only apply to environments using basic health reporting. See Health colors and statuses for details related to enhanced health.

Elastic Load Balancing health checks

In a load-balanced environment, Elastic Load Balancing sends a request to each instance in an environment every 10 seconds to confirm that instances are healthy. By default, the load balancer is configured to open a TCP connection on port 80. If the instance acknowledges the connection, it is considered healthy.

You can choose to override this setting by specifying an existing resource in your application. If you specify a path, such as `/health`, the health check URL is set to `HTTP:80/health`. The health check URL should be set to a path that is always served by your application. If it is set to a static page that is served or cached by the web server in front of your application, health checks will not reveal issues with the application server or web container. For instructions on modifying your health check URL, see Health check.

If a health check URL is configured, Elastic Load Balancing expects a GET request that it sends to return a response of `200 OK`. The application fails the health check if it fails to respond within 5 seconds or if it responds with any other HTTP status code. After 5 consecutive health check failures, Elastic Load Balancing takes the instance out of service.

For more information about Elastic Load Balancing health checks, see Health Check in the *Elastic Load Balancing User Guide*.

Note

Configuring a health check URL does not change the health check behavior of an environment's Auto Scaling group. An unhealthy instance is removed from the load balancer, but is not automatically replaced by Amazon EC2 Auto Scaling unless you configure Amazon EC2 Auto Scaling to use the Elastic Load Balancing health check as a basis for replacing instances. To configure Amazon EC2 Auto Scaling to replace instances that fail an Elastic Load Balancing health check, see Auto Scaling health check setting for your Elastic Beanstalk environment.

Single instance and worker tier environment health checks

In a single instance or worker tier environment, Elastic Beanstalk determines the instance's health by monitoring its Amazon EC2 instance status. Elastic Load Balancing health settings, including HTTP health check URLs, cannot be used in these environment types.

For more information on Amazon EC2 instance status checks, see Monitoring Instances with Status Checks in the *Amazon EC2 User Guide*.

Additional checks

In addition to Elastic Load Balancing health checks, Elastic Beanstalk monitors resources in your environment and changes health status to red if they fail to deploy, are not configured correctly, or become unavailable. These checks confirm that:

- The environment's Auto Scaling group is available and has a minimum of at least one instance.
- The environment's security group is available and is configured to allow incoming traffic on port 80.
- The environment CNAME exists and is pointing to the right load balancer.
- In a worker environment, the Amazon Simple Queue Service (Amazon SQS) queue is being polled at least once every three minutes.

Amazon CloudWatch metrics

With basic health reporting, the Elastic Beanstalk service does not publish any metrics to Amazon CloudWatch. The CloudWatch metrics used to produce graphs on the Monitoring page of the environment console are published by the resources in your environment.

For example, EC2 publishes the following metrics for the instances in your environment's Auto Scaling group:

`CPUUtilization`

Percentage of compute units currently in use.

`DiskReadBytes`, `DiskReadOps`, `DiskWriteBytes`, `DiskWriteOps`

Number of bytes read and written, and number of read and write operations.

`NetworkIn`, `NetworkOut`

Number of bytes sent and received.

Elastic Load Balancing publishes the following metrics for your environment's load balancer:

`BackendConnectionErrors`

Number of connection failures between the load balancer and environment instances.

`HTTPCode_Backend_2XX`, `HTTPCode_Backend_4XX`

Number of successful (2XX) and client error (4XX) response codes generated by instances in your environment.

`Latency`

Number of seconds between when the load balancer relays a request to an instance and when the response is received.

`RequestCount`

Number of completed requests.

These lists are not comprehensive. For a full list of metrics that can be reported for these resources, see the following topics in the Amazon CloudWatch Developer Guide:

Metrics

Namespace	Topic
AWS::ElasticLoadBalancing::LoadBalancer	Elastic Load Balancing Metrics and Resources
AWS::AutoScaling::AutoScalingGroup	Amazon Elastic Compute Cloud Metrics and Resources
AWS::SQS::Queue	Amazon SQS Metrics and Resources
AWS::RDS::DBInstance	Amazon RDS Dimensions and Metrics

Worker environment health metric

For worker environments only, the SQS daemon publishes a custom metric for environment health to CloudWatch, where a value of 1 is Green. You can review the CloudWatch health metric data in your account using the `ElasticBeanstalk/SQSD` namespace. The metric dimension is `EnvironmentName`, and the metric name is `Health`. All instances publish their metrics to the same namespace.

To enable the daemon to publish metrics, the environment's instance profile must have permission to call `cloudwatch:PutMetricData`. This permission is included in the default instance profile. For more information, see Managing Elastic Beanstalk instance profiles.

Enhanced health reporting and monitoring in Elastic Beanstalk

This section explains the functionality of the Elastic Beanstalk Enhanced Health feature.

Enhanced health reporting is a feature that you can enable on your environment to allow AWS Elastic Beanstalk to gather additional information about resources in your environment. Elastic Beanstalk analyzes the information gathered to provide a better picture of overall environment health and aid in the identification of issues that can cause your application to become unavailable.

In addition to changes in how health color works, enhanced health adds a *status* descriptor that provides an indicator of the severity of issues observed when an environment is yellow or red. When more information is available about the current status, you can choose the **Causes** button to view detailed health information on the health page.

To provide detailed health information about the Amazon EC2 instances running in your environment, Elastic Beanstalk includes a health agent in the Amazon Machine Image (AMI) for each platform version that supports enhanced health. The health agent monitors web server logs and system metrics and relays them to the Elastic Beanstalk service. Elastic Beanstalk analyzes these metrics and data from Elastic Load Balancing and Amazon EC2 Auto Scaling to provide an overall picture of an environment's health.

In addition to collecting and presenting information about your environment's resources, Elastic Beanstalk monitors the resources in your environment for several error conditions and provides notifications to help you avoid failures and resolve configuration issues. Factors that influence your environment's health include the results of each request served by your application, metrics from your instances' operating system, and the status of the most recent deployment.

You can view health status in real time by using the environment overview page of the Elastic Beanstalk console or the eb health command in the Elastic Beanstalk command line interface (EB CLI). To record and track environment and instance health over time, you can configure your environment to publish the information gathered by Elastic Beanstalk for enhanced health reporting to Amazon CloudWatch as custom metrics. CloudWatch charges for custom metrics apply to all metrics other than `EnvironmentHealth`, which is free of charge.

Windows platform notes

When you enable enhanced health reporting on a Windows Server environment, don't change IIS logging configuration. For enhanced health monitoring to work correctly, IIS logging must be configured with the **W3C** format and the **ETW event only** or **Both log file and ETW event** log event destinations.

In addition, don't disable or stop the Elastic Beanstalk health agent Windows service on any of your environment's instances. To collect and report enhanced health information on an instance, this service should be enabled and running.

The first time you create an environment Elastic Beanstalk prompts you to create the required roles and enables enhanced health reporting by default. Continue reading for details on how enhanced health reporting works, or see Enabling Elastic Beanstalk enhanced health reporting to get started using it right away.

Topics

- Configuring enhanced health rules for an environment
- Publishing Amazon CloudWatch custom metrics for an environment
- Using enhanced health reporting with the Elastic Beanstalk API
- Enhanced health log format
- Notifications and troubleshooting

The Elastic Beanstalk health agent

The Elastic Beanstalk health agent is a daemon process (or service, on Windows environments) that runs on each Amazon EC2 instance in your environment, monitoring operating system and application-level health metrics and reporting issues to Elastic Beanstalk. The health agent is included in all platform versions starting with version 2.0 of each platform.

The health agent reports similar metrics to those published to CloudWatch by Amazon EC2 Auto Scaling and Elastic Load Balancing as part of basic health reporting, including CPU load, HTTP codes, and latency. The health agent, however, reports directly to Elastic Beanstalk, with greater granularity and frequency than basic health reporting.

For basic health, these metrics are published every five minutes and can be monitored with graphs in the environment management console. With enhanced health, the Elastic Beanstalk health agent reports metrics to Elastic Beanstalk every 10 seconds. Elastic Beanstalk uses the metrics provided by the health agent to determine the health status of each instance in the environment and, combined with other factors, to determine the overall health of the environment.

The overall health of the environment can be viewed in real time in the environment overview page of the Elastic Beanstalk console, and is published to CloudWatch by Elastic Beanstalk every 60 seconds. You can view detailed metrics reported by the health agent in real time with the **eb health** command in the EB CLI.

For an additional charge, you can choose to publish individual instance and environment-level metrics to CloudWatch every 60 seconds. Metrics published to CloudWatch can then be used to create monitoring graphs in the environment management console.

Enhanced health reporting only incurs a charge if you choose to publish enhanced health metrics to CloudWatch. When you use enhanced health, you still get the basic health metrics published for free, even if you don't choose to publish enhanced health metrics.

See Instance metrics for details on the metrics reported by the health agent. For details on publishing enhanced health metrics to CloudWatch, see Publishing Amazon CloudWatch custom metrics for an environment.

Factors in determining instance and environment health

In addition to the basic health reporting system checks, including Elastic Load Balancing health checks and resource monitoring, Elastic Beanstalk enhanced health reporting gathers additional data about the state of the instances in your environment. This includes operating system metrics, server logs, and the state of ongoing environment operations such as deployments and updates. The Elastic Beanstalk health reporting service combines information from all available sources and analyzes it to determine the overall health of the environment.

Operations and commands

When you perform an operation on your environment, such as deploying a new version of an application, Elastic Beanstalk makes several changes that affect the environment's health status.

For example, when you deploy a new version of an application to an environment that is running multiple instances, you might see messages similar to the following as you monitor the environment's health with the EB CLI.

```
 id             status     cause
   Overall      Info       Command is executing on 3 out of 5 instances
 i-bb65c145     Pending    91 % of CPU is in use. 24 % in I/O wait
                           Performing application deployment (running for 31 seconds)
 i-ba65c144     Pending    Performing initialization (running for 12 seconds)
 i-f6a2d525     Ok         Application deployment completed 23 seconds ago and took 26
seconds
 i-e8a2d53b     Pending    94 % of CPU is in use. 52 % in I/O wait
                           Performing application deployment (running for 33 seconds)
 i-e81cca40     Ok
```

In this example, the overall status of the environment is Ok and the cause of this status is that the *Command is executing on 3 out of 5 instances*. Three of the instances in the environment have the status *Pending*, indicating that an operation is in progress.

When an operation completes, Elastic Beanstalk reports additional information about the operation. For the example, Elastic Beanstalk displays the following information about an instance that has already been updated with the new version of the application:

```
i-f6a2d525     Ok        Application deployment completed 23 seconds ago and took 26
 seconds
```

Instance health information also includes details about the most recent deployment to each instance in your environment. Each instance reports a deployment ID and status. The deployment ID is an integer that increases by one each time you deploy a new version of your application or change settings for on-instance configuration options, such as environment variables. You can use the deployment information to identify instances that are running the wrong version of your application after a failed rolling deployment.

In the cause column, Elastic Beanstalk includes informational messages about successful operations and other healthy states across multiple health checks, but they don't persist indefinitely. Causes for unhealthy environment statuses persist until the environment returns to a healthy status.

Command timeout

Elastic Beanstalk applies a command timeout from the time an operation begins to allow an instance to transition into a healthy state. This command timeout is set in your environment's update and deployment configuration (in the aws:elasticbeanstalk:command namespace) and defaults to 10 minutes.

During rolling updates, Elastic Beanstalk applies a separate timeout to each batch in the operation. This timeout is set as part of the environment's rolling update configuration (in the aws:autoscaling:updatepolicy:rollingupdate namespace). If all instances in the batch are healthy within the rolling update timeout, the operation continues to the next batch. If not, the operation fails.

Note

If your application does not pass health checks with an **OK** status but is stable at a different level, you can set the `HealthCheckSuccessThreshold` option in the `aws:elasticbeanstalk:command namespace` to change the level at which Elastic Beanstalk considers an instance to be healthy.

For a web server environment to be considered healthy, each instance in the environment or batch must pass 12 consecutive health checks over the course of two minutes. For a worker tier environment, each instance must pass 18 health checks. Before the command times out, Elastic

Beanstalk doesn't lower an environment's health status when health checks fail. If the instances in the environment become healthy within the command timeout, the operation succeeds.

HTTP requests

When no operation is in progress on an environment, the primary source of information about instance and environment health is the web server logs for each instance. To determine the health of an instance and the overall health of the environment, Elastic Beanstalk considers the number of requests, the result of each request, and the speed at which each request was resolved.

On Linux-based platforms, Elastic Beanstalk reads and parses web server logs to get information about HTTP requests. On the Windows Server platform, Elastic Beanstalk receives this information directly from the IIS web server.

Your environment might not have an active web server. For example, the Multicontainer Docker platform doesn't include a web server. Other platforms include a web server, and your application might disable it. In these cases, your environment requires additional configuration to provide the Elastic Beanstalk health agent with logs in the format that it needs to relay health information to the Elastic Beanstalk service. See Enhanced health log format for details.

Operating system metrics

Elastic Beanstalk monitors operating system metrics reported by the health agent to identify instances that are consistently low on system resources.

See Instance metrics for details on the metrics reported by the health agent.

Health check rule customization

Elastic Beanstalk enhanced health reporting relies on a set of rules to determine the health of your environment. Some of these rules might not be appropriate for your particular application. A common case is an application that returns frequent HTTP 4xx errors by design. Elastic Beanstalk, using one of its default rules, concludes that something is going wrong, and changes your environment health status from OK to Warning, Degraded, or Severe, depending on the error rate. To handle this case correctly, Elastic Beanstalk allows you to configure this rule and ignore application HTTP 4xx errors. For details, see Configuring enhanced health rules for an environment.

Enhanced health roles

Enhanced health reporting requires two roles—a service role for Elastic Beanstalk and an instance profile for the environment. The service role allows Elastic Beanstalk to interact with other AWS services on your behalf to gather information about the resources in your environment. The instance profile allows the instances in your environment to write logs to Amazon S3 and to communicate enhanced health information to the Elastic Beanstalk service.

When you create an Elastic Beanstalk environment using the Elastic Beanstalk console or the EB CLI, Elastic Beanstalk creates a default service role and attaches required managed policies to a default instance profile for your environment.

If you use the API, an SDK, or the AWS CLI to create environments, you must create these roles in advance, and specify them during environment creation to use enhanced health. For instructions on creating appropriate roles for your environments, see Elastic Beanstalk Service roles, instance profiles, and user policies.

We recommend that you use *managed policies* for your instance profile and service role. Managed policies are AWS Identity and Access Management (IAM) policies that Elastic Beanstalk maintains. Using managed policies guarantees that your environment has all permissions it needs to function properly.

For the instance profile, you can use the `AWSElasticBeanstalkWebTier` or `AWSElasticBeanstalkWorkerTier` managed policy, for a web server tier or worker tier environment, respectively. For details about these two managed instance profile policies, see the section called “Instance profiles”.

Enhanced health authorization

The Elastic Beanstalk instance profile managed policies contain permissions for the `elasticbeanstalk:PutInstanceStatistics` action. This action isn't part of the Elastic Beanstalk API. It's part of a different API that environment instances use internally to communicate enhanced health information to the Elastic Beanstalk service. You don't call this API directly.

When you create a new environment, authorization for the `elasticbeanstalk:PutInstanceStatistics` action is enabled by default. To increase security of your environment and help prevent health data spoofing on your behalf, we recommend that you keep authorization for this action enabled. If you use managed policies for your instance profile, this feature is available for your new environment without any further configuration. However, If you use a *custom instance profile* instead of a *managed policy*, your environment might display a **No Data** health status. This happens because the instances aren't authorized for the action that communicates enhanced health data to the service.

To authorize the action, include the following statement in your instance profile.

```
{
  "Sid": "ElasticBeanstalkHealthAccess",
  "Action": [
    "elasticbeanstalk:PutInstanceStatistics"
  ],
  "Effect": "Allow",
  "Resource": [
    "arn:aws:elasticbeanstalk:*:*:application/*",
    "arn:aws:elasticbeanstalk:*:*:environment/*"
  ]
}
```

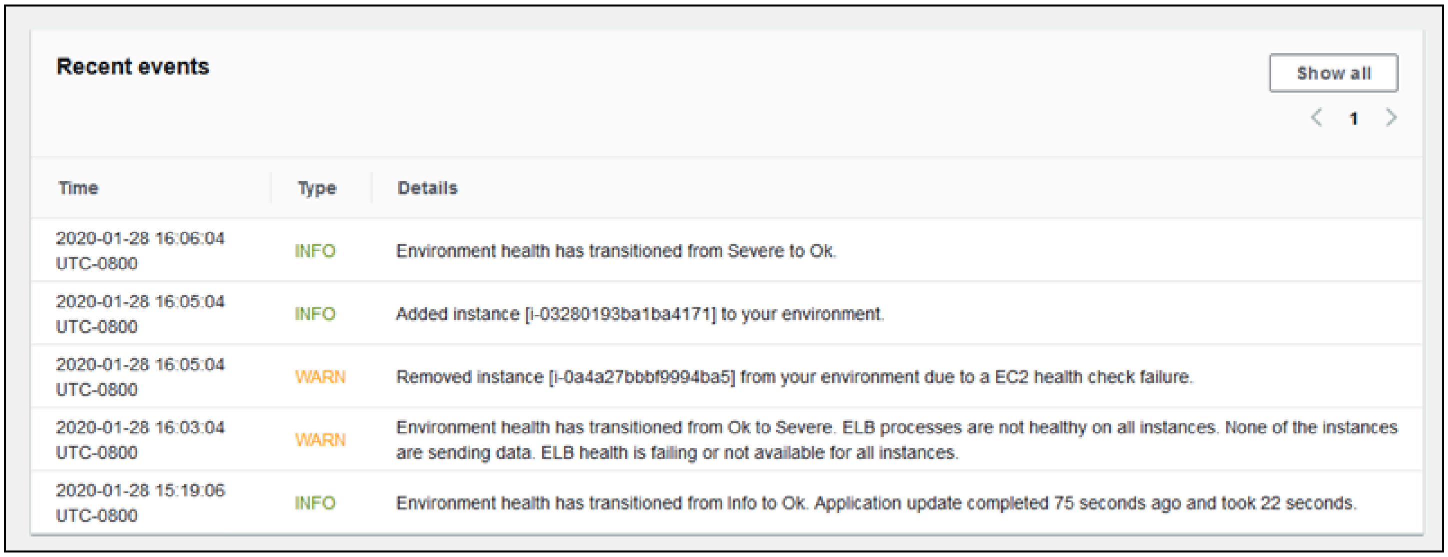
If you don’t want to use enhanced health authorization at this time, disable it by setting set the `EnhancedHealthAuthEnabled` option in the the section called “aws:elasticbeanstalk:healthreporting:system” namespace to `false`. If this option is disabled, the permissions described previously aren’t required. You can remove them from the instance profile for least privilege access to your applications and environments.

Note

Previously the default setting for `EnhancedHealthAuthEnabled` was `false`, which resulted in authorization for the `elasticbeanstalk:PutInstanceStatistics` action also being disabled by default. To enable this action for an existing environment, set the `EnhancedHealthAuthEnabled` option in the the section called “aws:elasticbeanstalk:healthreporting:system” namespace to `true`. You can configure this option by using an option setting in a configuration file.

Enhanced health events

The enhanced health system generates events when an environment transitions between states. The following example shows events output by an environment transitioning between **Info**, **OK**, and **Severe** states.


Recent events

Show all

Time	Type	Details
2020-01-28 16:06:04 UTC-0800	INFO	Environment health has transitioned from Severe to Ok.
2020-01-28 16:05:04 UTC-0800	INFO	Added instance [i-03280193ba1ba4171] to your environment.
2020-01-28 16:05:04 UTC-0800	WARN	Removed instance [i-0a4a27bbbf9994ba5] from your environment due to a EC2 health check failure.
2020-01-28 16:03:04 UTC-0800	WARN	Environment health has transitioned from Ok to Severe. ELB processes are not healthy on all instances. None of the instances are sending data. ELB health is failing or not available for all instances.
2020-01-28 15:19:06 UTC-0800	INFO	Environment health has transitioned from Info to Ok. Application update completed 75 seconds ago and took 22 seconds.

When transitioning to a worse state, the enhanced health event includes a message indicating the transition cause.

Not all changes in status at an instance level cause Elastic Beanstalk to emit an event. To prevent false alarms, Elastic Beanstalk generates a health-related event only if an issue persists across multiple checks.

Real-time environment-level health information, including status, color, and cause, is available in the environment overview page of the Elastic Beanstalk console and the EB CLI. By attaching the EB CLI to your environment and running the **eb health** command, you can also view real-time statuses from each of the instances in your environment.

Enhanced health reporting behavior during updates, deployments, and scaling

Enabling enhanced health reporting can affect how your environment behaves during configuration updates and deployments. Elastic Beanstalk won't complete a batch of updates until all of the instances pass health checks consistently. Also, because enhanced health reporting applies a higher standard for health and monitors more factors, instances that pass basic health reporting's ELB health check won't necessarily pass with enhanced health reporting. See the topics on rolling configuration updates and rolling deployments for details on how health checks affect the update process.

Enhanced health reporting can also highlight the need to set a proper health check URL for Elastic Load Balancing. When your environment scales up to meet demand, new instances will start taking requests as soon as they pass enough ELB health checks. If a health check URL is not configured, this can be as little as 20 seconds after a new instance is able to accept a TCP connection.

If your application hasn't finished starting up by the time the load balancer declares it healthy enough to receive traffic, you will see a flood of failed requests, and your environment will start to fail health checks. A health check URL that hits a path served by your application can prevent this issue. ELB health checks won't pass until a GET request to the health check URL returns a 200 status code.

Enabling Elastic Beanstalk enhanced health reporting

This topic explains how enhanced health reporting is enabled. It provides procedures for you to enable the enhanced health feature for your environment with the Elastic Beanstalk console, the EB CLI, and with an .ebextensions configuration.

New environments created with the latest platform versions include the AWS Elastic Beanstalk health agent, which supports enhanced health reporting. If you create your environment in the Elastic Beanstalk console or with the EB CLI, enhanced health is enabled by default. You can also set your health reporting preference in your application's source code using configuration files.

Enhanced health reporting requires an instance profile and service role with the standard set of permissions. When you create an environment in the Elastic Beanstalk console, Elastic Beanstalk creates the required roles automatically. See Learn how to get started with Elastic Beanstalk for instructions on creating your first environment.

Topics

Enabling enhanced health reporting using the Elastic Beanstalk console

To enable enhanced health reporting in a running environment using the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

3. In the navigation pane, choose **Configuration**.
4. In the **Monitoring** configuration category, choose **Edit**.
5. Under **Health reporting**, for **System**, choose **Enhanced**.

   Note

   The options for enhanced health reporting don't appear if you are using an unsupported platform or version.
6. To save the changes choose **Apply** at the bottom of the page.

The Elastic Beanstalk console defaults to enhanced health reporting when you create a new environment with a version 2 (v2) platform version. You can disable enhanced health reporting by changing the health reporting option during environment creation.

To disable enhanced health reporting when creating an environment using the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. Create an application or select an existing one.
3. Create an environment. On the **Create a new environment** page, before choosing **Create environment**, choose **Configure more options**.
4. In the **Monitoring** configuration category, choose **Edit**.
5. Under **Health reporting**, for **System**, choose **Basic**.
6. Choose **Save**.

Enabling enhanced health reporting using the EB CLI

When you create a new environment with the **eb create** command, the EB CLI enables enhanced health reporting by default and applies the default instance profile and service role.

You can specify a different service role by name by using the `--service-role` option.

If you have an environment running with basic health reporting on a v2 platform version and you want to switch to enhanced health, follow these steps.

To enable enhanced health on a running environment using the EB CLI

1. Use the **eb config** command to open the configuration file in the default text editor.

```
~/project$ eb config
```

2. Locate the `aws:elasticbeanstalk:environment` namespace in the settings section. Ensure that the value of `ServiceRole` is not null and that it matches the name of your service role.

```
  aws:elasticbeanstalk:environment:
    EnvironmentType: LoadBalanced
    ServiceRole: aws-elasticbeanstalk-service-role
```

3. Under the `aws:elasticbeanstalk:healthreporting:system:` namespace, change the value of `SystemType` to **enhanced**.

```
  aws:elasticbeanstalk:healthreporting:system:
    SystemType: enhanced
```

4. Save the configuration file and close the text editor.
5. The EB CLI starts an environment update to apply your configuration changes. Wait for the operation to complete or press **Ctrl+C** to exit safely.

```
~/project$ eb config
Printing Status:
INFO: Environment update is starting.
INFO: Health reporting type changed to ENHANCED.
INFO: Updating environment no-role-test's configuration settings.
```

Enabling enhanced health reporting using a configuration file

You can enable enhanced health reporting by including a configuration file in your source bundle. The following example shows a configuration file that enables enhanced health reporting and assigns the default service and instance profile to the environment:

Example .ebextensions/enhanced-health.config

```
option_settings:
  aws:elasticbeanstalk:healthreporting:system:
    SystemType: enhanced
  aws:autoscaling:launchconfiguration:
    IamInstanceProfile: aws-elasticbeanstalk-ec2-role
  aws:elasticbeanstalk:environment:
    ServiceRole: aws-elasticbeanstalk-service-role
```

If you created your own instance profile or service role, replace the highlighted text with the names of those roles.

Enhanced health monitoring with the environment management console

When you enable enhanced health reporting in AWS Elastic Beanstalk, you can monitor environment health in the environment management console.

Topics

- Environment overview
- Environment health page
- Monitoring page

Environment overview

The environment overview displays the health status of the environment and lists events that provide information about recent changes in health status.

To view the environment overview

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

For detailed information about the current environment's health, open the **Health** page by choosing **Causes**. Alternatively, in the navigation pane, choose **Health**.

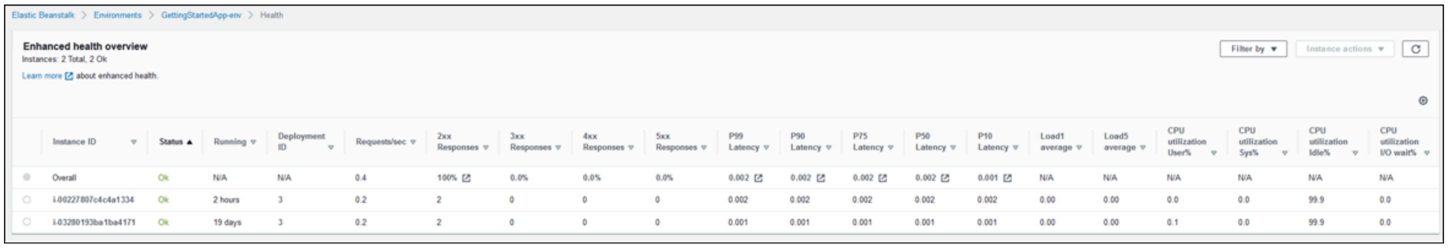
Environment health page

The **Health** page displays health status, metrics, and causes for the environment and for each Amazon EC2 instance in the environment.

Note

Elastic Beanstalk displays the **Health** page only if you have enabled enhanced health monitoring for the environment.

The following image shows the **Health** page for a Linux environment.

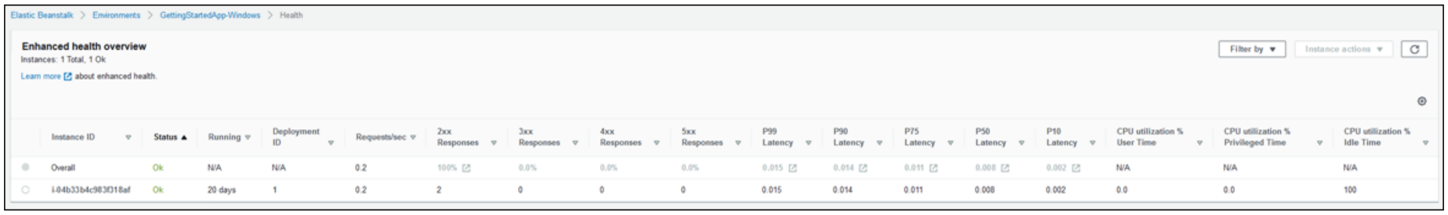


Elastic Beanstalk > Environments > GettingStartedApp-env > Health

Enhanced health overview
Instances: 2 Total, 2 Ok
Learn more about enhanced health.

Instance ID	Status	Running	Deployment ID	Requests/sec	2xx Responses	3xx Responses	4xx Responses	5xx Responses	P99 Latency	P90 Latency	P75 Latency	P50 Latency	P10 Latency	Load1 average	Load5 average	CPU utilization User%	CPU utilization Sys%	CPU utilization Idle%	CPU utilization I/O wait%
Overall	Ok	N/A	N/A	0.4	100%	0.0%	0.0%	0.0%	0.002	0.002	0.002	0.002	0.001	N/A	N/A	N/A	N/A	N/A	N/A
i-00227807c4c4a1334	Ok	2 hours	3	0.2	2	0	0	0	0.002	0.002	0.002	0.002	0.002	0.00	0.00	0.0	0.0	99.9	0.0
i-03280193ba1ba4171	Ok	19 days	3	0.2	2	0	0	0	0.001	0.001	0.001	0.001	0.001	0.00	0.00	0.1	0.0	99.9	0.0

The following image shows the **Health** page for a Windows environment. Notice that CPU metrics are different from those on a Linux environment.

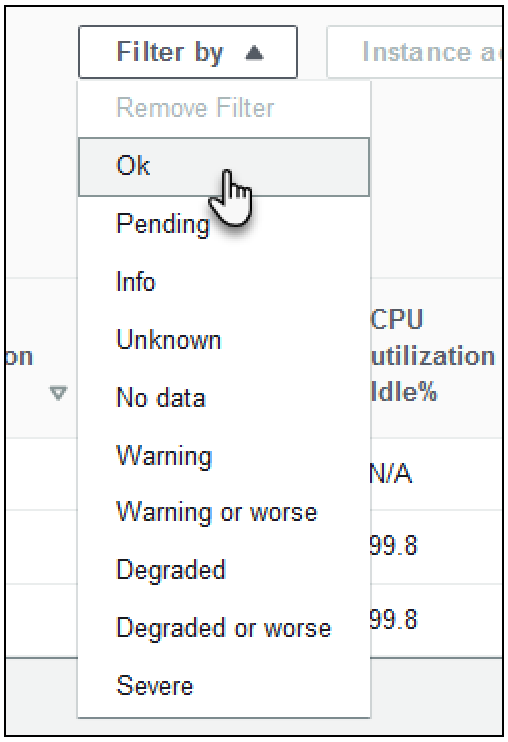


Elastic Beanstalk > Environments > GettingStartedApp-Windows > Health

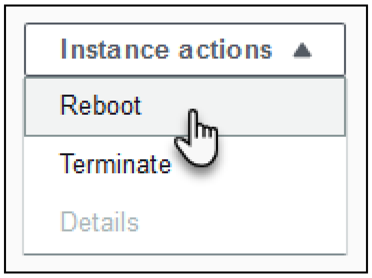
Enhanced health overview
Instances: 1 Total, 1 Ok
Learn more about enhanced health.

Instance ID	Status	Running	Deployment ID	Requests/sec	2xx Responses	3xx Responses	4xx Responses	5xx Responses	P99 Latency	P90 Latency	P75 Latency	P50 Latency	P10 Latency	CPU utilization % User Time	CPU utilization % Privileged Time	CPU utilization % Idle Time
Overall	Ok	N/A	N/A	0.2	100%	0.0%	0.0%	0.0%	0.015	0.014	0.011	0.008	0.002	N/A	N/A	N/A
i-04b33b4c983f318af	Ok	20 days	1	0.2	2	0	0	0	0.015	0.014	0.011	0.008	0.002	0.0	0.0	100

At the top of the page you can see the total number of environment instances, as well as the number of instances per status. To display only instances that have a particular status, choose**Filter By**, and then select a status.



To reboot or terminate an unhealthy instance, choose **Instance Actions**, and then choose **Reboot** or **Terminate**.

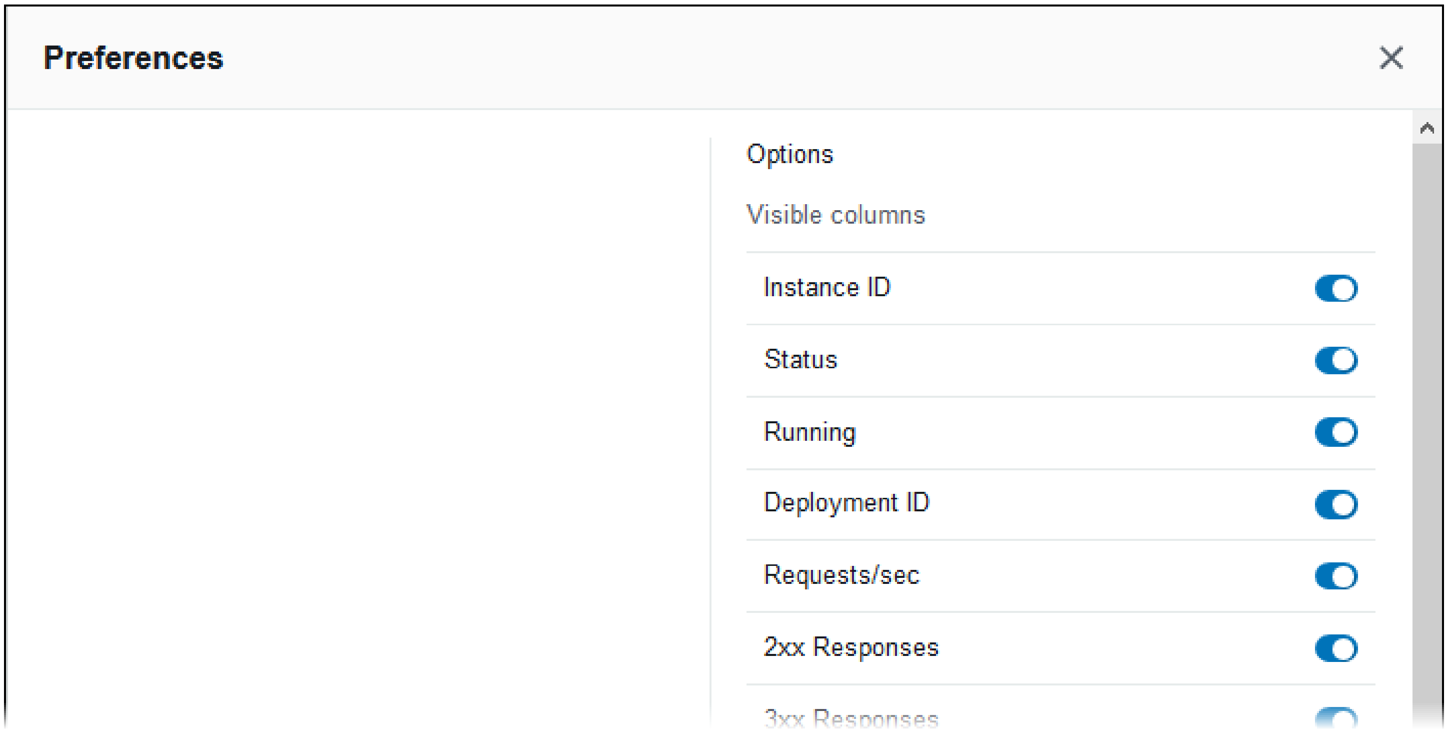


Elastic Beanstalk updates the **Health** page every 10 seconds. It reports information about environment and instance health.

For each Amazon EC2 instance in the environment, the page displays the instance's ID and status, the amount of time since the instance was launched, the ID of the most recent deployment executed on the instance, the responses and latency of requests that the instance served, and load and CPU utilization information. The **Overall** row displays average response and latency information for the entire environment.

The page displays many details in a very wide table. To hide some of the columns, choose

(**Preferences**). Select or clear column names, and then choose **Confirm**.



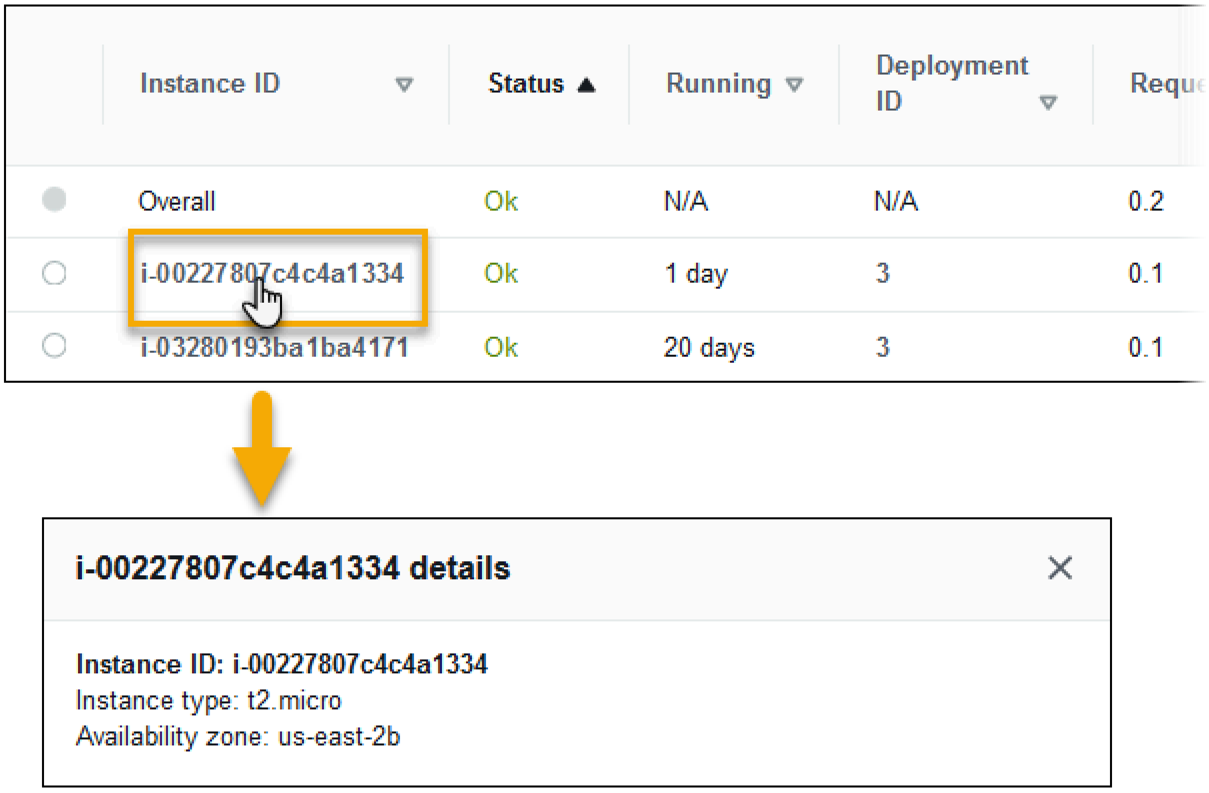
Preferences

Options

Visible columns

Instance ID	●
Status	●
Running	●
Deployment ID	●
Requests/sec	●
2xx Responses	●
3xx Responses	●

Choose the **Instance ID** of any instance to view more information about the instance, including its Availability Zone and instance type.

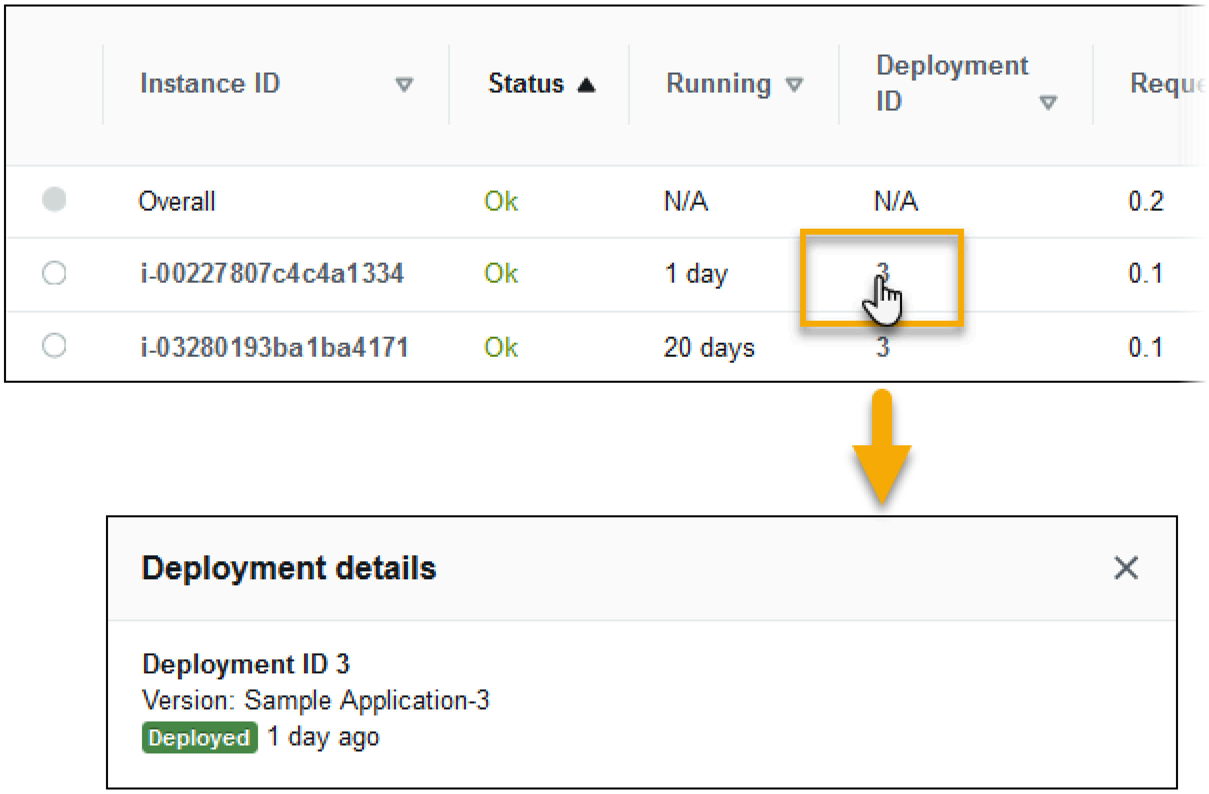


	Instance ID	Status ▲	Running	Deployment ID	Reque
●	Overall	Ok	N/A	N/A	0.2
○	i-00227807c4c4a1334	Ok	1 day	3	0.1
○	i-03280193ba1ba4171	Ok	20 days	3	0.1

i-00227807c4c4a1334 details

Instance ID: i-00227807c4c4a1334
Instance type: t2.micro
Availability zone: us-east-2b

Choose the **Deployment ID** of any instance to view information about the last deployment to the instance.



	Instance ID	Status	Running	Deployment ID	Reque
●	Overall	Ok	N/A	N/A	0.2
○	i-00227807c4c4a1334	Ok	1 day	3	0.1
○	i-03280193ba1ba4171	Ok	20 days	3	0.1

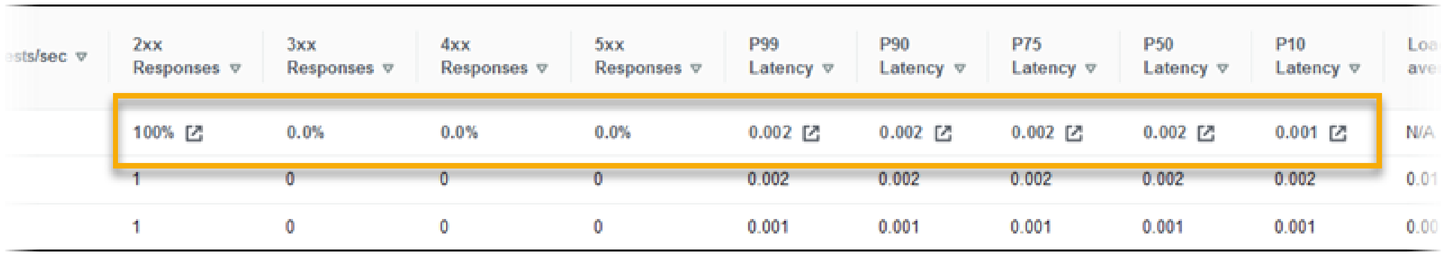
Deployment details

Deployment ID 3
Version: Sample Application-3
Deployed 1 day ago

Deployment information includes the following:

- **Deployment ID**—The unique identifier for the deployment. Deployment IDs starts at 1 and increase by one each time you deploy a new application version or change configuration settings that affect the software or operating system running on the instances in your environment.
- **Version**—The version label of the application source code used in the deployment.
- **Status**—The status of the deployment, which can be `In Progress`, `Deployed`, or `Failed`.
- **Time**— For in-progress deployments, the time that the deployment started. For completed deployments, the time that the deployment ended.

If you enable X-Ray integration on your environment and instrument your application with the AWS X-Ray SDK, the **Health** page adds links to the AWS X-Ray console in the overview row.



sts/sec	2xx Responses	3xx Responses	4xx Responses	5xx Responses	P99 Latency	P90 Latency	P75 Latency	P50 Latency	P10 Latency	Loa ave
	100%	0.0%	0.0%	0.0%	0.002	0.002	0.002	0.002	0.001	N/A
	1	0	0	0	0.002	0.002	0.002	0.002	0.002	0.01
	1	0	0	0	0.001	0.001	0.001	0.001	0.001	0.00

Choose a link to view traces related to the highlighted statistic in the AWS X-Ray console.

Monitoring page

The **Monitoring** page displays summary statistics and graphs for the custom Amazon CloudWatch metrics generated by the enhanced health reporting system. See Monitoring environment health in the AWS management console for instructions on adding graphs and statistics to this page.

Health colors and statuses

Enhanced health reporting represents instance and overall environment health by using four colors, similar to basic health reporting. Enhanced health reporting also provides seven health statuses, which are single-word descriptors that provide a better indication of the state of your environment.

Instance status and environment status

Every time Elastic Beanstalk runs a health check on your environment, enhanced health reporting checks the health of each instance in your environment by analyzing all of the data available. If any lower-level check fails, Elastic Beanstalk downgrades the health of the instance.

Elastic Beanstalk displays the health information for the overall environment (color, status, and cause) in the environment management console. This information is also available in the EB CLI. Health status and cause messages for individual instances are updated every 10 seconds and are available from the EB CLI when you view health status with **eb health**.

Elastic Beanstalk uses changes in instance health to evaluate environment health, but does not immediately change environment health status. When an instance fails health checks at least three times in any one-minute period, Elastic Beanstalk may downgrade the health of the environment. Depending on the number of instances in the environment and the issue identified, one unhealthy instance can cause Elastic Beanstalk to display an informational message or to change the environment's health status from green (**OK**) to yellow (**Warning**) or red (**Degraded** or **Severe**).

OK (green)

This status is displayed when:

- An instance is passing health checks and the health agent is not reporting any problems.
- Most instances in the environment are passing health checks and the health agent is not reporting major issues.
- An instance is passing health checks and is completing requests normally.

Example: Your environment was recently deployed and is taking requests normally. Five percent of requests are returning 400 series errors. Deployment completed normally on each instance.

Message (instance): Application deployment completed 23 seconds ago and took 26 seconds.

Warning (yellow)

This status is displayed when:

- The health agent is reporting a moderate number of request failures or other issues for an instance or environment.
- An operation is in progress on an instance and is taking a very long time.

Example: One instance in the environment has a status of **Severe**.

Message (environment): Impaired services on 1 out of 5 instances.

Degraded (red)

This status is displayed when the health agent is reporting a high number of request failures or other issues for an instance or environment.

Example: environment is in the process of scaling up to 5 instances.

Message (environment): 4 active instances is below Auto Scaling group minimum size 5.

Severe (red)

This status is displayed when the health agent is reporting a very high number of request failures or other issues for an instance or environment.

Example: Elastic Beanstalk is unable to contact the load balancer to get instance health.

Message (environment): ELB health is failing or not available for all instances. None of the instances are sending data. Unable to assume role "arn:aws:iam::123456789012:role/aws-elasticbeanstalk-service-role". Verify that the role exists and is configured correctly.

Message (Instances): Instance ELB health has not been available for 37 minutes. No data. Last seen 37 minutes ago.

Info (green)

This status is displayed when:

- An operation is in progress on an instance.
- An operation is in progress on several instances in an environment.

Example: A new application version is being deployed to running instances.

Message (environment): Command is executing on 3 out of 5 instances.

Message (instance): Performing application deployment (running for 3 seconds).

Pending (grey)

This status is displayed when an operation is in progress on an instance within the command timeout.

Example: You have recently created the environment and instances are being bootstrapped.

Message: Performing initialization (running for 12 seconds).

Unknown (grey)

This status is displayed when Elastic Beanstalk and the health agent are reporting an insufficient amount of data on an instance.

Example: No data is being received.

Suspended (grey)

This status is displayed when Elastic Beanstalk stopped monitoring the environment's health. The environment might not work correctly. Some severe health conditions, if they last a long time, cause Elastic Beanstalk to transition the environment to the **Suspended** status.

Example: Elastic Beanstalk can't access the environment's service role.

Example: The Auto Scaling group that Elastic Beanstalk created for the environment has been deleted.

Message: Environment health has transitioned from **OK** to **Severe**. There are no instances. Auto Scaling group desired capacity is set to 1.

Instance metrics

Instance metrics provide information about the health of instances in your environment. The Elastic Beanstalk health agent runs on each instance. It gathers and relays metrics about instances to Elastic Beanstalk, which analyzes the metrics to determine the health of the instances in your environments.

The on-instance Elastic Beanstalk health agent gathers metrics about instances from web servers and the operating system. To get web server information on Linux-based platforms, Elastic Beanstalk reads and parses web server logs. On the Windows Server platform, Elastic Beanstalk receives this information directly from the IIS web server. Web servers provide information about incoming HTTP requests: how many requests came in, how many resulted in errors, and how long they took to resolve. The operating system provides snapshot information about the state of the instances' resources: the CPU load and distribution of time spent on each process type.

The health agent gathers web server and operating system metrics and relays them to Elastic Beanstalk every 10 seconds. Elastic Beanstalk analyzes the data and uses the results to update the health status for each instance and the environment.

Topics

Web server metrics

On Linux-based platforms, the Elastic Beanstalk health agent reads web server metrics from logs generated by the web container or server that processes requests on each instance in your environment. Elastic Beanstalk platforms are configured to generate two logs: one in human-readable format and one in machine-readable format. The health agent relays machine-readable logs to Elastic Beanstalk every 10 seconds.

For more information on the log format used by Elastic Beanstalk, see Enhanced health log format.

On the Windows Server platform, Elastic Beanstalk adds a module to the IIS web server's request pipeline and captures metrics about HTTP request times and response codes. The module sends these metrics to the on-instance health agent using a high-performance interprocess communication (IPC) channel. For implementation details, see Web server metrics capture in IIS on Windows server.

Reported Web Server Metrics

`RequestCount`

Number of requests handled by the web server per second over the last 10 seconds. Shown as an average `r/sec` (requests per second) in the EB CLI and Environment health page.

`Status2xx`, `Status3xx`, `Status4xx`, `Status5xx`

Number of requests that resulted in each type of status code over the last 10 seconds. For example, successful requests return a 200 OK, redirects are a 301, and a 404 is returned if the URL entered doesn't match any resources in the application.

The EB CLI and Environment health page show these metrics both as a raw number of requests for instances, and as a percentage of overall requests for environments.

`p99.9`, `p99`, `p95`, `p90`, `p85`, `p75`, `p50`, `p10`

Average latency for the slowest *x* percent of requests over the last 10 seconds, where *x* is the difference between the number and 100. For example, `p99 1.403` indicates the slowest 1% of requests over the last 10 seconds had an average latency of 1.403 seconds.

Operating system metrics

The Elastic Beanstalk health agent reports the following operating system metrics. Elastic Beanstalk uses these metrics to identify instances that are under sustained heavy load. The metrics differ by operating system.

Reported operating system metrics—Linux

`Running`

The amount of time that has passed since the instance was launched.

`Load 1`, `Load 5`

Load average in the last one-minute and five-minute periods. Shown as a decimal value indicating the average number of processes running during that time. If the number shown is higher than the number of vCPUs (threads) available, then the remainder is the average number of processes that were waiting.

For example, if your instance type has four vCPUs, and the load is 4.5, there was an average of .5 processes in wait during that time period, equivalent to one process waiting 50 percent of the time.

`User %`, `Nice %`, `System %`, `Idle %`, `I/O Wait %`

Percentage of time that the CPU has spent in each state over the last 10 seconds.

Reported operating system metrics—Windows

`Running`

The amount of time that has passed since the instance was launched.

`% User Time`, `% Privileged Time`, `% Idle Time`

Percentage of time that the CPU has spent in each state over the last 10 seconds.

Web server metrics capture in IIS on Windows server

On the Windows Server platform, Elastic Beanstalk adds a module to the IIS web server's request pipeline and captures metrics about HTTP request times and response codes. The module sends these metrics to the on-instance health agent using a high-performance interprocess communication (IPC) channel. The health agent aggregates these metrics, combines them with operating system metrics, and sends them to the Elastic Beanstalk service.

Implementation details

To capture metrics from IIS, Elastic Beanstalk implements a managed `IHttpModule`, and subscribes to the `BeginRequest` and `EndRequest` events. This enables the module to report HTTP request latency and response codes for all web requests handled by IIS. To add the module to the IIS request pipeline, Elastic Beanstalk registers the module in the `<modules>` section of the IIS configuration file, `%windir%\System32\inetsrv\config\applicationHost.config`.

The Elastic Beanstalk module in IIS sends the captured web request metrics to the on-instance health agent, which is a Windows service named `HealthD`. To send this data, the module uses `NetNamedPipeBinding`, which provides a secure and reliable binding that is optimized for on-machine communication.

Configuring enhanced health rules for an environment

AWS Elastic Beanstalk enhanced health reporting relies on a set of rules to determine the health of your environment. Some of these rules might not be appropriate for your particular application. The following are some common examples:

- You use client-side test tools. In this case, frequent HTTP client (4xx) errors are expected.
- You use AWS WAF in conjunction with your environment's Application Load Balancer to block unwanted incoming traffic. In this case, Application Load Balancer returns HTTP 403 for each rejected incoming message.

By default, Elastic Beanstalk includes all application HTTP 4xx errors when determining the environment's health. It changes your environment health status from **OK** to **Warning**, **Degraded**, or **Severe**, depending on the error rate. To correctly handle cases such as the examples we mentioned, Elastic Beanstalk enables you to configure some enhanced health rules. You can choose to ignore application HTTP 4xx errors on the environment's instances, or to ignore HTTP 4xx errors returned by the environment's load balancer. This topic describes how to make these configuration changes.

Note

Currently, these are the only available enhanced heath rule customizations. You can't configure enhanced health to ignore other HTTP errors in addition to 4xx.

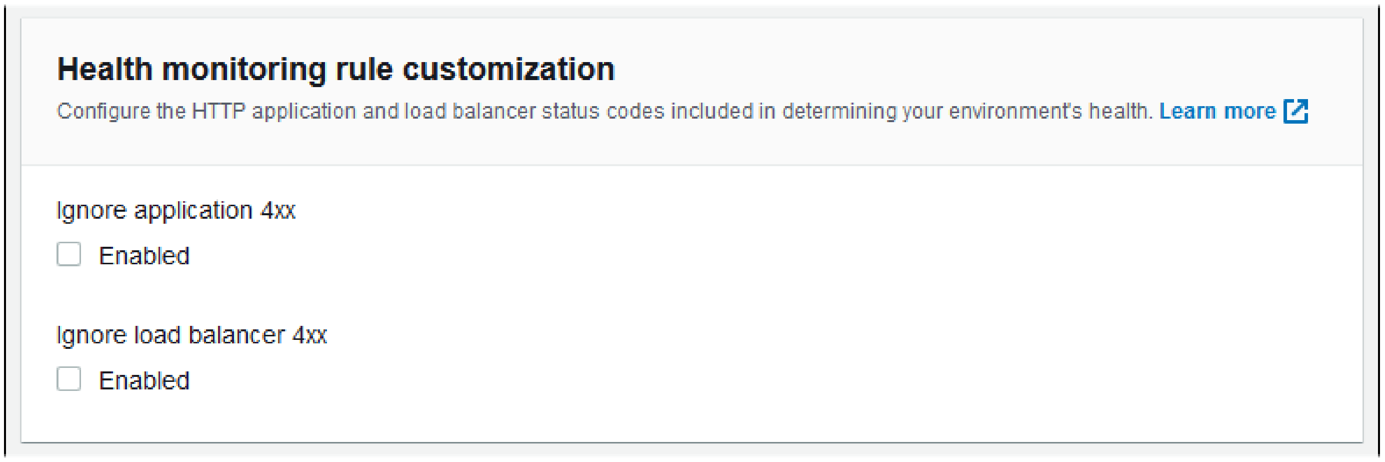
Configuring enhanced health rules using the Elastic Beanstalk console

You can use the Elastic Beanstalk console to configure enhanced health rules in your environment.

To configure HTTP 4xx status code checking using the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Monitoring** configuration category, choose **Edit**.
5. Under **Health monitoring rule customization**, enable or disable the desired **Ignore** options.


Health monitoring rule customization

Configure the HTTP application and load balancer status codes included in determining your environment's health. Learn more

Ignore application 4xx

☐ Enabled

Ignore load balancer 4xx

☐ Enabled

6. To save the changes choose **Apply** at the bottom of the page.

Configuring enhanced health rules using the EB CLI

You can use the EB CLI to configure enhanced health rules by saving your environment's configuration locally, adding an entry that configures enhanced health rules, and then uploading the configuration to Elastic Beanstalk. You can apply the saved configuration to an environment during or after creation.

To configure HTTP 4xx status code checking using the EB CLI and saved configurations

1. Initialize your project folder with [eb init].
2. Create an environment by running the **eb create** command.
3. Save a configuration template locally by running the **eb config save** command. The following example uses the `--cfg` option to specify the name of the configuration.

```
$ eb config save --cfg 01-base-state
Configuration saved at: ~/project/.elasticbeanstalk/saved_configs/01-base-
state.cfg.yml
```

4. Open the saved configuration file in a text editor.

5. Under `OptionSettings` > `aws:elasticbeanstalk:healthreporting:system:`, add a `ConfigDocument` key to list each enhanced health rule to configure. The following `ConfigDocument` disables the checking of application HTTP 4xx status codes, while keeping the checking of load balancer HTTP 4xx code enabled.

```
OptionSettings:
  ...
  aws:elasticbeanstalk:healthreporting:system:
    ConfigDocument:
      Rules:
        Environment:
          Application:
            ApplicationRequests4xx:
              Enabled: false
          ELB:
            ELBRequests4xx:
              Enabled: true
      Version: 1
    SystemType: enhanced
...
```

Note

You can combine `Rules` and `CloudWatchMetrics` in the same `ConfigDocument` option setting. `CloudWatchMetrics` are described in Publishing Amazon CloudWatch custom metrics for an environment.

If you previously enabled `CloudWatchMetrics`, the configuration file that you retrieve using the **eb config save** command already has a `ConfigDocument` key with a `CloudWatchMetrics` section. *Do not delete it*—add a `Rules` section into the same `ConfigDocument` option value.

6. Save the configuration file and close the text editor. In this example, the updated configuration file is saved with a name (`02-cloudwatch-enabled.cfg.yml`) that's different from the downloaded configuration file. This creates a separate saved configuration when the file is uploaded. You can use the same name as the downloaded file to overwrite the existing configuration without creating a new one.

7. Use the **eb config put** command to upload the updated configuration file to Elastic Beanstalk.

```
$ eb config put 02-cloudwatch-enabled
```

When using the **eb config** `get` and `put` commands with saved configurations, don't include the file name extension.

8. Apply the saved configuration to your running environment.

```
$ eb config --cfg 02-cloudwatch-enabled
```

The `--cfg` option specifies a named configuration file that is applied to the environment. You can save the configuration file locally or in Elastic Beanstalk. If a configuration file with the specified name exists in both locations, the EB CLI uses the local file.

Configuring enhanced health rules using a config document

The configuration (config) document for enhanced health rules is a JSON document that lists the rules to configure.

The following example shows a config document that disables the checking of application HTTP 4xx status codes and enables the checking of load balancer HTTP 4xx status codes.

```
{
  "Rules": {
    "Environment": {
      "Application": {
        "ApplicationRequests4xx": {
          "Enabled": false
        }
      },
      "ELB": {
        "ELBRequests4xx": {
          "Enabled": true
        }
      }
    }
  },
  "Version": 1
}
```

For the AWS CLI, you pass the document as a value for the `Value` key in an option settings argument, which itself is a JSON object. In this case, you must escape quotation marks in the embedded document. The following command checks if the configuration settings are valid.

```
$ aws elasticbeanstalk validate-configuration-settings --application-name my-app --
environment-name my-env --option-settings '[
    {
        "Namespace": "aws:elasticbeanstalk:healthreporting:system",
        "OptionName": "ConfigDocument",
        "Value": "{\"Rules\": { \"Environment\": { \"Application\":
 { \"ApplicationRequests4xx\": { \"Enabled\": false } }, \"ELB\": { \"ELBRequests4xx\":
 {\"Enabled\": true } } } }, \"Version\": 1 }"
    }
]'
```

For an `.ebextensions` configuration file in YAML, you can provide the JSON document as is.

```
  option_settings:
    - namespace: aws:elasticbeanstalk:healthreporting:system
      option_name: ConfigDocument
      value: {
  "Rules": {
    "Environment": {
      "Application": {
        "ApplicationRequests4xx": {
          "Enabled": false
        }
      },
      "ELB": {
        "ELBRequests4xx": {
          "Enabled": true
        }
      }
    }
  },
  "Version": 1
}
```

Publishing Amazon CloudWatch custom metrics for an environment

You can publish the data gathered by AWS Elastic Beanstalk enhanced health reporting to Amazon CloudWatch as custom metrics. Publishing metrics to CloudWatch lets you monitor changes in application performance over time and identify potential issues by tracking how resource usage and request latency scale with load.

By publishing metrics to CloudWatch, you also make them available for use with monitoring graphs and alarms. One free metric, *EnvironmentHealth*, is enabled automatically when you use enhanced health reporting. Custom metrics other than *EnvironmentHealth* incur standard CloudWatch charges.

To publish CloudWatch custom metrics for an environment, you must first enable enhanced health reporting on the environment. See Enabling Elastic Beanstalk enhanced health reporting for instructions.

Topics

Enhanced health reporting metrics

When you enable enhanced health reporting in your environment, the enhanced health reporting system automatically publishes one CloudWatch custom metric, *EnvironmentHealth*. To publish additional metrics to CloudWatch, configure your environment with those metrics by using the Elastic Beanstalk console, EB CLI, or .ebextensions.

You can publish the following enhanced health metrics from your environment to CloudWatch.

Available metrics—all platforms

`EnvironmentHealth`

Environment only. This is the only CloudWatch metric that the enhanced health reporting system publishes, unless you configure additional metrics. Environment health is represented by one of seven statuses. In the CloudWatch console, these statuses map to the following values:

- 0 – OK
- 1 – Info
- 5 – Unknown

- 10 – No data
- 15 – Warning
- 20 – Degraded
- 25 – Severe

`InstancesSevere`, `InstancesDegraded`, `InstancesWarning`, `InstancesInfo`, `InstancesOk`, `InstancesPending`, `InstancesUnknown`, `InstancesNoData`

Environment only. These metrics indicate the number of instances in the environment with each health status. `InstancesNoData` indicates the number of instances for which no data is being received.

`ApplicationRequestsTotal`, `ApplicationRequests5xx`, `ApplicationRequests4xx`, `ApplicationRequests3xx`, `ApplicationRequests2xx`

Instance and environment. Indicates the total number of requests completed by the instance or environment, and the number of requests that completed with each status code category.

`ApplicationLatencyP10`, `ApplicationLatencyP50`, `ApplicationLatencyP75`, `ApplicationLatencyP85`, `ApplicationLatencyP90`, `ApplicationLatencyP95`, `ApplicationLatencyP99`, `ApplicationLatencyP99.9`

Instance and environment. Indicates the average amount of time, in seconds, it takes to complete the fastest *x* percent of requests.

`InstanceHealth`

Instance only. Indicates the current health status of the instance. Instance health is represented by one of seven statuses. In the CloudWatch console, these statuses map to the following values:

- 0 – OK
- 1 – Info
- 5 – Unknown
- 10 – No data
- 15 – Warning
- 20 – Degraded
- 25 – Severe

Available metrics—Linux

`CPUIrq`, `CPUIdle`, `CPUUser`, `CPUSystem`, `CPUSoftirq`, `CPUIowait`, `CPUNice`

Instance only. Indicates the percentage of time that the CPU has spent in each state over the last minute.

`LoadAverage1min`

Instance only. The average CPU load of the instance over the last minute.

`RootFilesystemUtil`

Instance only. Indicates the percentage of disk space that's in use.

Available metrics—Windows

`CPUIdle`, `CPUUser`, `CPUPrivileged`

Instance only. Indicates the percentage of time that the CPU has spent in each state over the last minute.

Configuring CloudWatch metrics using the Elastic Beanstalk console

You can use the Elastic Beanstalk console to configure your environment to publish enhanced health reporting metrics to CloudWatch and make them available for use with monitoring graphs and alarms.

To configure CloudWatch custom metrics in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Monitoring** configuration category, choose **Edit**.
5. Under **Health reporting**, select the instance and environment metrics to publish to CloudWatch. To select multiple metrics, press the **Ctrl** key while choosing.
6. To save the changes choose **Apply** at the bottom of the page.

Enabling CloudWatch custom metrics adds them to the list of metrics available on the Monitoring page.

Configuring CloudWatch custom metrics using the EB CLI

You can use the EB CLI to configure custom metrics by saving your environment's configuration locally, adding an entry that defines the metrics to publish, and then uploading the configuration to Elastic Beanstalk. You can apply the saved configuration to an environment during or after creation.

To configure CloudWatch custom metrics with the EB CLI and saved configurations

1. Initialize your project folder with **eb init**.
2. Create an environment by running the **eb create** command.
3. Save a configuration template locally by running the **eb config save** command. The following example uses the `--cfg` option to specify the name of the configuration.

```
$ eb config save --cfg 01-base-state
Configuration saved at: ~/project/.elasticbeanstalk/saved_configs/01-base-
state.cfg.yml
```

4. Open the saved configuration file in a text editor.
5. Under `OptionSettings` > `aws:elasticbeanstalk:healthreporting:system:`, add a `ConfigDocument` key to enable each of the CloudWatch metrics you want. For example, the following `ConfigDocument` publishes `ApplicationRequests5xx` and `ApplicationRequests4xx` metrics at the environment level, and `ApplicationRequestsTotal` metrics at the instance level.

```
OptionSettings:
  ...
  aws:elasticbeanstalk:healthreporting:system:
    ConfigDocument:
      CloudWatchMetrics:
        Environment:
          ApplicationRequests5xx: 60
          ApplicationRequests4xx: 60
        Instance:
          ApplicationRequestsTotal: 60
      Version: 1
    SystemType: enhanced
```

```
...
```

In the example, 60 indicates the number of seconds between measurements. Currently, this is the only supported value.

> **Note**
>
> You can combine `CloudWatchMetrics` and `Rules` in the same `ConfigDocument` option setting. `Rules` are described in Configuring enhanced health rules for an environment.
>
> If you previously used `Rules` to configure enhanced health rules, then the configuration file that you retrieve using the **eb config save** command already has a `ConfigDocument` key with a `Rules` section. *Do not delete it*—add a `CloudWatchMetrics` section into the same `ConfigDocument` option value.

6. Save the configuration file and close the text editor. In this example, the updated configuration file is saved with a name (`02-cloudwatch-enabled.cfg.yml`) that is different from the downloaded configuration file. This creates a separate saved configuration when the file is uploaded. You can use the same name as the downloaded file to overwrite the existing configuration without creating a new one.
7. Use the **eb config put** command to upload the updated configuration file to Elastic Beanstalk.

```
$ eb config put 02-cloudwatch-enabled
```

When using the **eb config** `get` and `put` commands with saved configurations, don't include the file extension.

8. Apply the saved configuration to your running environment.

```
$ eb config --cfg 02-cloudwatch-enabled
```

The `--cfg` option specifies a named configuration file that is applied to the environment. You can save the configuration file locally or in Elastic Beanstalk. If a configuration file with the specified name exists in both locations, the EB CLI uses the local file.

Providing custom metric config documents

The configuration (config) document for Amazon CloudWatch custom metrics is a JSON document that lists the metrics to publish at the environment and instance levels. The following example shows a config document that enables all custom metrics available on Linux.

```
{
  "CloudWatchMetrics": {
    "Environment": {
      "ApplicationLatencyP99.9": 60,
      "InstancesSevere": 60,
      "ApplicationLatencyP90": 60,
      "ApplicationLatencyP99": 60,
      "ApplicationLatencyP95": 60,
      "InstancesUnknown": 60,
      "ApplicationLatencyP85": 60,
      "InstancesInfo": 60,
      "ApplicationRequests2xx": 60,
      "InstancesDegraded": 60,
      "InstancesWarning": 60,
      "ApplicationLatencyP50": 60,
      "ApplicationRequestsTotal": 60,
      "InstancesNoData": 60,
      "InstancesPending": 60,
      "ApplicationLatencyP10": 60,
      "ApplicationRequests5xx": 60,
      "ApplicationLatencyP75": 60,
      "InstancesOk": 60,
      "ApplicationRequests3xx": 60,
      "ApplicationRequests4xx": 60
    },
    "Instance": {
      "ApplicationLatencyP99.9": 60,
      "ApplicationLatencyP90": 60,
      "ApplicationLatencyP99": 60,
      "ApplicationLatencyP95": 60,
      "ApplicationLatencyP85": 60,
      "CPUUser": 60,
      "ApplicationRequests2xx": 60,
      "CPUIdle": 60,
      "ApplicationLatencyP50": 60,
      "ApplicationRequestsTotal": 60,
      "RootFilesystemUtil": 60,
      "LoadAverage1min": 60,
      "CPUIrq": 60,
      "CPUNice": 60,
      "CPUIowait": 60,
      "ApplicationLatencyP10": 60,
      "LoadAverage5min": 60,
      "ApplicationRequests5xx": 60,
      "ApplicationLatencyP75": 60,
      "CPUSystem": 60,
      "ApplicationRequests3xx": 60,
      "ApplicationRequests4xx": 60,
      "InstanceHealth": 60,
      "CPUSoftirq": 60
    }
  },
  "Version": 1
}
```

For the AWS CLI, you pass the document as a value for the `Value` key in an option settings argument, which itself is a JSON object. In this case, you must escape quotation marks in the embedded document.

```
$ aws elasticbeanstalk validate-configuration-settings --application-name my-app --
environment-name my-env --option-settings '[
    {
        "Namespace": "aws:elasticbeanstalk:healthreporting:system",
        "OptionName": "ConfigDocument",
        "Value": "{\"CloudWatchMetrics\": {\"Environment\":
 {\"ApplicationLatencyP99.9\": 60,\"InstancesSevere\": 60,\"ApplicationLatencyP90\":
 60,\"ApplicationLatencyP99\": 60,\"ApplicationLatencyP95\": 60,\"InstancesUnknown
\": 60,\"ApplicationLatencyP85\": 60,\"InstancesInfo\": 60,\"ApplicationRequests2xx
\": 60,\"InstancesDegraded\": 60,\"InstancesWarning\": 60,\"ApplicationLatencyP50\":
 60,\"ApplicationRequestsTotal\": 60,\"InstancesNoData\": 60,\"InstancesPending
\": 60,\"ApplicationLatencyP10\": 60,\"ApplicationRequests5xx\": 60,
\"ApplicationLatencyP75\": 60,\"InstancesOk\": 60,\"ApplicationRequests3xx\": 60,
\"ApplicationRequests4xx\": 60},\"Instance\": {\"ApplicationLatencyP99.9\": 60,
\"ApplicationLatencyP90\": 60,\"ApplicationLatencyP99\": 60,\"ApplicationLatencyP95\":
 60,\"ApplicationLatencyP85\": 60,\"CPUUser\": 60,\"ApplicationRequests2xx\":
 60,\"CPUIdle\": 60,\"ApplicationLatencyP50\": 60,\"ApplicationRequestsTotal\":
 60,\"RootFilesystemUtil\": 60,\"LoadAverage1min\": 60,\"CPUIrq\": 60,\"CPUNice
\": 60,\"CPUIowait\": 60,\"ApplicationLatencyP10\": 60,\"LoadAverage5min\": 60,
\"ApplicationRequests5xx\": 60,\"ApplicationLatencyP75\": 60,\"CPUSystem\": 60,
\"ApplicationRequests3xx\": 60,\"ApplicationRequests4xx\": 60,\"InstanceHealth\": 60,
\"CPUSoftirq\": 60}},\"Version\": 1}"
    }
]'
```

For an `.ebextensions` configuration file in YAML, you can provide the JSON document as is.

```
  option_settings:
    - namespace: aws:elasticbeanstalk:healthreporting:system
      option_name: ConfigDocument
      value: {
  "CloudWatchMetrics": {
    "Environment": {
      "ApplicationLatencyP99.9": 60,
      "InstancesSevere": 60,
      "ApplicationLatencyP90": 60,
      "ApplicationLatencyP99": 60,
      "ApplicationLatencyP95": 60,
      "InstancesUnknown": 60,
      "ApplicationLatencyP85": 60,
      "InstancesInfo": 60,
      "ApplicationRequests2xx": 60,
      "InstancesDegraded": 60,
      "InstancesWarning": 60,
      "ApplicationLatencyP50": 60,
      "ApplicationRequestsTotal": 60,
      "InstancesNoData": 60,
      "InstancesPending": 60,
      "ApplicationLatencyP10": 60,
      "ApplicationRequests5xx": 60,
      "ApplicationLatencyP75": 60,
      "InstancesOk": 60,
      "ApplicationRequests3xx": 60,
      "ApplicationRequests4xx": 60
    },
    "Instance": {
      "ApplicationLatencyP99.9": 60,
      "ApplicationLatencyP90": 60,
      "ApplicationLatencyP99": 60,
      "ApplicationLatencyP95": 60,
      "ApplicationLatencyP85": 60,
      "CPUUser": 60,
      "ApplicationRequests2xx": 60,
      "CPUIdle": 60,
      "ApplicationLatencyP50": 60,
      "ApplicationRequestsTotal": 60,
      "RootFilesystemUtil": 60,
      "LoadAverage1min": 60,
      "CPUIrq": 60,
      "CPUNice": 60,
      "CPUIowait": 60,
      "ApplicationLatencyP10": 60,
      "LoadAverage5min": 60,
      "ApplicationRequests5xx": 60,
      "ApplicationLatencyP75": 60,
      "CPUSystem": 60,
      "ApplicationRequests3xx": 60,
      "ApplicationRequests4xx": 60,
      "InstanceHealth": 60,
      "CPUSoftirq": 60
    }
  },
  "Version": 1
}
```

Using enhanced health reporting with the Elastic Beanstalk API

Because AWS Elastic Beanstalk enhanced health reporting has role and solution stack requirements, you must update scripts and code that you used prior to the release of enhanced health reporting before you can use it. To maintain backward compatibility, enhanced health reporting is not enabled by default when you create an environment using the Elastic Beanstalk API.

You configure enhanced health reporting by setting the service role, the instance profile, and Amazon CloudWatch configuration options for your environment. You can do this in three ways: by setting the configuration options in the `.ebextensions` folder, with saved configurations, or by configuring them directly in the `create-environment` call's `option-settings` parameter.

To use the API, SDKs, or AWS command line interface (CLI) to create an environment that supports enhanced health, you must:

- Create a service role and instance profile with the appropriate permissions
- Create a new environment with a new platform version
- Set the health system type, instance profile, and service role configuration options

Use the following configuration options in the `aws:elasticbeanstalk:healthreporting:system`, `aws:autoscaling:launchconfiguration`, and `aws:elasticbeanstalk:environment` namespaces to configure your environment for enhanced health reporting.

Enhanced health configuration options

SystemType

Namespace: `aws:elasticbeanstalk:healthreporting:system`

To enable enhanced health reporting, set to **enhanced**.

IamInstanceProfile

Namespace: `aws:autoscaling:launchconfiguration`

Set to the name of an instance profile configured for use with Elastic Beanstalk.

ServiceRole

Namespace: `aws:elasticbeanstalk:environment`

Set to the name of a service role configured for use with Elastic Beanstalk.

ConfigDocument (optional)

Namespace: `aws:elasticbeanstalk:healthreporting:system`

A JSON document that defines the and instance and environment metrics to publish to CloudWatch. For example:

```
{
  "CloudWatchMetrics":
    {
    "Environment":
      {
      "ApplicationLatencyP99.9":60,
      "InstancesSevere":60
      }
    "Instance":
      {
      "ApplicationLatencyP85":60,
      "CPUUser": 60
      }
    }
  "Version":1
}
```

Note

Config documents may require special formatting, such as escaping quotes, depending on how you provide them to Elastic Beanstalk. See Providing custom metric config documents for examples.

Enhanced health log format

AWS Elastic Beanstalk platforms use a custom web server log format to efficiently relay information about HTTP requests to the enhanced health reporting system. The system analyzes the logs, identifies issues, and sets the instance and environment health accordingly. If you disable the web server proxy on your environment and serve requests directly from the web container, you can still make full use of enhanced health reporting by configuring your server to output logs in the location and format that the Elastic Beanstalk health agent uses.

Note

The information on this page is relevant only to Linux-based platforms. On the Windows Server platform, Elastic Beanstalk receives information about HTTP requests directly from the IIS web server. For details, see Web server metrics capture in IIS on Windows server.

Web server log configuration

Elastic Beanstalk platforms are configured to output two logs with information about HTTP requests. The first is in verbose format and provides detailed information about the request, including the requester's user agent information and a human-readable timestamp.

/var/log/nginx/access.log

The following example is from an nginx proxy running on a Ruby web server environment, but the format is similar for Apache.

```
172.31.24.3 - - [23/Jul/2015:00:21:20 +0000] "GET / HTTP/1.1" 200 11 "-" "curl/7.22.0
 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23
 librtmp/2.3" "177.72.242.17"
172.31.24.3 - - [23/Jul/2015:00:21:21 +0000] "GET / HTTP/1.1" 200 11 "-" "curl/7.22.0
 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23
 librtmp/2.3" "177.72.242.17"
172.31.24.3 - - [23/Jul/2015:00:21:22 +0000] "GET / HTTP/1.1" 200 11 "-" "curl/7.22.0
 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23
 librtmp/2.3" "177.72.242.17"
172.31.24.3 - - [23/Jul/2015:00:21:22 +0000] "GET / HTTP/1.1" 200 11 "-" "curl/7.22.0
 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23
 librtmp/2.3" "177.72.242.17"
172.31.24.3 - - [23/Jul/2015:00:21:22 +0000] "GET / HTTP/1.1" 200 11 "-" "curl/7.22.0
 (x86_64-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23
 librtmp/2.3" "177.72.242.17"
```

The second log is in terse format. It includes information relevant only to enhanced health reporting. This log is output to a subfolder named `healthd` and rotates hourly. Old logs are deleted immediately after rotating out.

/var/log/nginx/healthd/application.log.2015-07-23-00

The following example shows a log in the machine-readable format.

```
1437609879.311"/"200"0.083"0.083"177.72.242.17
1437609879.874"/"200"0.347"0.347"177.72.242.17
1437609880.006"/bad/path"404"0.001"0.001"177.72.242.17
1437609880.058"/"200"0.530"0.530"177.72.242.17
1437609880.928"/bad/path"404"0.001"0.001"177.72.242.17
```

The enhanced health log format includes the following information:

- The time of the request, in Unix time
- The path of the request
- The HTTP status code for the result
- The request time
- The upstream time
- The `X-Forwarded-For` HTTP header

For nginx proxies, times are printed in floating-point seconds, with three decimal places. For Apache, whole microseconds are used.

Note

If you see a warning similar to the following in a log file, where `DATE-TIME` is a date and time, and you are using a custom proxy, such as in a multi-container Docker environment, you must use an .ebextension to configure your environment so that `healthd` can read your log files:

```
W, [DATE-TIME #1922] WARN -- : log file "/var/log/nginx/healthd/
application.log.DATE-TIME" does not exist
```

You can start with the .ebextension in the Multicontainer Docker sample.

/etc/nginx/conf.d/webapp_healthd.conf

The following example shows the log configuration for nginx with the `healthd` log format highlighted.

```
upstream my_app {
  server unix:///var/run/puma/my_app.sock;
}

log_format healthd '$msec"$uri"'
                '$status"$request_time"$upstream_response_time"'
                '$http_x_forwarded_for';

server {
  listen 80;
  server_name _ localhost; # need to listen to localhost for worker tier

  if ($time_iso8601 ~ "^(\d{4})-(\d{2})-(\d{2})T(\d{2})") {
    set $year $1;
    set $month $2;
    set $day $3;
    set $hour $4;
  }

  access_log  /var/log/nginx/access.log  main;
  access_log /var/log/nginx/healthd/application.log.$year-$month-$day-$hour healthd;

  location / {
    proxy_pass http://my_app; # match the name of upstream directive which is defined
 above
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  }

  location /assets {
    alias /var/app/current/public/assets;
    gzip_static on;
    gzip on;
    expires max;
    add_header Cache-Control public;
  }

  location /public {
    alias /var/app/current/public;
    gzip_static on;
    gzip on;
    expires max;
    add_header Cache-Control public;
  }
}
```

/etc/httpd/conf.d/healthd.conf

The following example shows the log configuration for Apache.

```
LogFormat "%{%s}t\"%U\"%s\"%D\"%D\"%{X-Forwarded-For}i" healthd
CustomLog "|/usr/sbin/rotatelogs /var/log/httpd/healthd/application.log.%Y-%m-%d-%H
 3600" healthd
```

Generating logs for enhanced health reporting

To provide logs to the health agent, you must do the following:

- Output logs in the correct format, as shown in the previous section
- Output logs to `/var/log/nginx/healthd/`
- Name logs using the following format: `application.log.$year-$month-$day-$hour`
- Rotate logs once per hour

- Do not truncate logs

Notifications and troubleshooting

This page lists example cause messages for common issues and links to more information. Cause messages appear in the environment overview page of the Elastic Beanstalk console and are recorded in events when health issues persist across several checks.

Deployments

Elastic Beanstalk monitors your environment for consistency following deployments. If a rolling deployment fails, the version of your application running on the instances in your environment may vary. This can occur if a deployment succeeds on one or more batches but fails prior to all batches completing.

Incorrect application version found on 2 out of 5 instances. Expected version "v1" (deployment 1).

Incorrect application version on environment instances. Expected version "v1" (deployment 1).

The expected application version is not running on some or all instances in an environment.

Incorrect application version "v2" (deployment 2). Expected version "v1" (deployment 1).

The application deployed to an instance differs from the expected version. If a deployment fails, the expected version is reset to the version from the most recent successful deployment. In the above example, the first deployment (version "v1") succeeded, but the second deployment (version "v2") failed. Any instances running "v2" are considered unhealthy.

To solve this issue, start another deployment. You can redeploy a previous version that you know works, or configure your environment to ignore health checks during deployment and redeploy the new version to force the deployment to complete.

You can also identify and terminate the instances that are running the wrong application version. Elastic Beanstalk will launch instances with the correct version to replace any instances that you terminate. Use the EB CLI health command to identify instances that are running the wrong application version.

Application server

15% of requests are erroring with HTTP 4xx

20% of the requests to the ELB are erroring with HTTP 4xx.

A high percentage of HTTP requests to an instance or environment are failing with 4xx errors.

A 400 series status code indicates that the user made a bad request, such as requesting a page that doesn't exist (404 File Not Found) or that the user doesn't have access to (403 Forbidden). A low number of 404s is not unusual but a large number could mean that there are internal or external links to unavailable pages. These issues can be resolved by fixing bad internal links and adding redirects for bad external links.

5% of the requests are failing with HTTP 5xx

3% of the requests to the ELB are failing with HTTP 5xx.

A high percentage of HTTP requests to an instance or environment are failing with 500 series status codes.

A 500 series status code indicates that the application server encountered an internal error. These issues indicate that there is an error in your application code and should be identified and fixed quickly.

95% of CPU is in use

On an instance, the health agent is reporting an extremely high percentage of CPU usage and sets the instance health to **Warning** or **Degraded**.

Scale your environment to take load off of instances.

Worker instance

20 messages waiting in the queue (25 seconds ago)

Requests are being added to your worker environment's queue faster than they can be processed. Scale your environment to increase capacity.

5 messages in Dead Letter Queue (15 seconds ago)

Worker requests are failing repeatedly and being added to the the section called "Dead-letter queues". Check the requests in the dead-letter queue to see why they are failing.

Other resources

4 active instances is below Auto Scaling group minimum size 5

The number of instances running in your environment is fewer than the minimum configured for the Auto Scaling group.

Auto Scaling group (groupname) notifications have been deleted or modified

The notifications configured for your Auto Scaling group have been modified outside of Elastic Beanstalk.

Manage alarms

This topic walks you through the steps to create alarms for metrics that you're monitoring. It also provides instructions to view your existing alarms and to check their state.

You can create alarms for metrics that you are monitoring by using the Elastic Beanstalk console. Alarms help you monitor changes to your AWS Elastic Beanstalk environment so that you can easily identify and mitigate problems before they occur. For example, you can set an alarm that notifies you when CPU utilization in an environment exceeds a certain threshold, ensuring that you are notified before a potential problem occurs. For more information, see Using Elastic Beanstalk with Amazon CloudWatch.

Note

Elastic Beanstalk uses CloudWatch for monitoring and alarms, meaning CloudWatch costs are applied to your AWS account for any alarms that you use.

For more information about monitoring specific metrics, see Basic health reporting.

To check the state of your alarms

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Alarms**.

   The page displays a list of existing alarms. If any alarms are in the alarm state, they are flagged with the warning icon (⚠).

4. To filter alarms, choose the drop-down menu, and then select a filter.
5. To edit or delete an alarm, choose the edit icon (⚙) or the delete icon (✖), respectively.

To create an alarm

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Monitoring**.
4. Locate the metric for which you want to create an alarm, and then choose the alarm icon (🔔). The **Add alarm** page is displayed.
5. Enter details about the alarm:
   - **Name**: A name for this alarm.
   - **Description** (optional): A short description of what this alarm is.
   - **Period**: The time interval between readings.
   - **Threshold**: Describes the behavior and value that the metric must exceed in order to trigger an alarm.
   - **Change state after**: The amount a time after a threshold has been exceed that triggers a change in state of the alarm.
   - **Notify**: The Amazon SNS topic that is notified when an alarm changes state.
   - **Notify when state changes to**:
     - **OK**: The metric is within the defined threshold.
     - **Alarm**: The metric exceeded the defined threshold.
     - **Insufficient data**: The alarm has just started, the metric is not available, or not enough data is available for the metric to determine the alarm state.
6. Choose **Add**. The environment status changes to gray while the environment updates. You can view the alarm that you created by choosing **Alarms** in the navigation pane.

Viewing an Elastic Beanstalk environment's change history

This topic explains how you can use the Elastic Beanstalk Console to view a history of configuration changes that have been made to your Elastic Beanstalk environments.

Elastic Beanstalk fetches your change history from events recorded in AWS CloudTrail and displays them in a list that you can easily navigate and filter.

The Change History panel displays the following information for changes made to your environments:

- The date and time when a change was made
- The IAM user that was responsible for a change made
- The source tool (either Elastic Beanstalk command line interface (EB CLI) or console) that was used to make the change
- The configuration parameter and new values that were set

Any sensitive data that is part of the change, such as the names of database users affected by the change, aren't displayed in the panel.

To view change history

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Change history**.

   The Change History page shows a list of configuration changes that were made to your Elastic Beanstalk environments.

Note the following points about navigating the information on this page:

- You can page through the list by choosing **<** (previous) or **>** (next), or by choosing a specific page number.
- Under the **Configuration changes** column, select the arrow icon to toggle between expanding and collapsing the list of changes under the **Changes made** heading.
- Use the search bar to filter your results from the change history list. You can enter any string to narrow down the list of changes that are displayed.

Note the following about filtering the displayed results:

- The search filter is not case sensitive.
- You can filter displayed changes based on information under the **Configuration changes** column, even when it is not visible due to being collapsed inside **Changes made**.
- You can only filter the results displayed. However, the filter remains in place even if you select to go to another page to display more results. Your filtered results also append to the result set of the next page.

The following examples demonstrate how the data shown on the earlier screen can be filtered:

- Enter **`GettingStartedApp-env`** in the search box to narrow down the results to only include the changes that were made to the environment named *GettingStartedApp-env*.
- Enter **`example3`** in the search box to narrow down the results to only include changes that were made by IAM users whose username contains the string *example3*.
- Enter **`2020-10`** in the search box to narrow down the results to only include changes that were made during the month of October 2020. Change the search value to **`2020-10-16`** to filter further the displayed results to only include changes that were made on the day of October 16, 2020.
- Enter **`proxy:staticfiles`** in the search box to narrow down the results to only include the changes that were made to the namespace named *aws:elasticbeanstalk:environment:proxy:staticfiles*. The rows that are displayed are the result of the filter. This is true even for results that are collapsed under **Changes made**.

Viewing an Elastic Beanstalk environment's event stream

This topic explains how to access events and notifications associated with your application.

Viewing events with the Elastic Beanstalk console

To view events with the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Events**.

The Events page shows a list of all events that have been recorded for the environment. You can page through the list choosing **<** (previous), **>** (next), or page numbers. You can filter the type of events shown by using the **Severity** drop-down list.

Viewing events with command line tools

The EB CLI and AWS CLI both provide commands for retrieving events. If you are managing your environment using the EB CLI, use **eb events** to print a list of events. This command also has a `--follow` option that continues to show new events until you press **Ctrl+C** to stop output.

To pull events using the AWS CLI, use the `describe-events` command and specify the environment by name or ID:

```
$ aws elasticbeanstalk describe-events --environment-id e-gbjzqccra3
{
    "Events": [
        {
            "ApplicationName": "elastic-beanstalk-example",
            "EnvironmentName": "elasticBeanstalkExa-env",
            "Severity": "INFO",
            "RequestId": "a4c7bfd6-2043-11e5-91e2-9114455c358a",
            "Message": "Environment update completed successfully.",
            "EventDate": "2015-07-01T22:52:12.639Z"
        },
...
```

For more information about the command line tools, see ???.

Listing and connecting to server instances

This topic explains how to view a list of the Amazon EC2 instances running your Elastic Beanstalk application environment and how to connect to them.

You can view a list of Amazon EC2 instances running your AWS Elastic Beanstalk application environment through the Elastic Beanstalk console. You can connect to the instances using any SSH client. You can connect to the instances running Windows using Remote Desktop.

Important

Before you can access your Elastic Beanstalk–provisioned Amazon EC2 instances, you must create an Amazon EC2 key pair and configure your Elastic Beanstalk–provisioned Amazon EC2instances to use the Amazon EC2 key pair. You can set up your Amazon EC2 key pairs using the AWS Management Console. For instructions on creating a key pair for Amazon EC2, see the *Amazon EC2 Getting Started Guide*. For more information on how to configure your Amazon EC2 instances to use an Amazon EC2 key pair, see EC2 key pair.
By default, Elastic Beanstalk does not enable remote connections to EC2 instances in a Windows container except for those in legacy Windows containers. (Elastic Beanstalk configures EC2 instances in legacy Windows containers to use port 3389 for RDP connections.) You can enable remote connections to your EC2 instances running Windows by adding a rule to a security group that authorizes inbound traffic to the instances. We strongly recommend that you remove the rule when you end your remote connection. You can add the rule again the next time you need to log in remotely. For more information, see Adding a Rule for Inbound RDP Traffic to a Windows Instance and Connect to Your Windows Instance in the *Amazon Elastic Compute Cloud User Guide for Microsoft Windows*.

To view and connect to Amazon EC2 instances for an environment

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. In the navigation pane of the console, choose **Load Balancers**.
3. Load balancers created by Elastic Beanstalk have **awseb** in the name. Find the load balancer for your environment and click it.
4. Choose the **Instances** tab in the bottom pane of the console.

   A list of the instances that the load balancer for your Elastic Beanstalk environment uses is displayed. Make a note of an instance ID that you want to connect to.
5. In the navigation pane of the Amazon EC2 console, choose **Instances**, and find your instance ID in the list.
6. Right-click the instance ID for the Amazon EC2 instance running in your environment's load balancer, and then select **Connect** from the context menu.
7. Make a note of the instance's public DNS address on the **Description** tab.
8. Connect to an instance running Linux by using the SSH client of your choice, and then type **ssh -i .ec2/mykeypair.pem ec2-user@<public-DNS-of-the-instance>** .

For more information on connecting to an Amazon EC2 Linux instance, see Getting Started with Amazon EC2 Linux Instances in the *Amazon EC2 User Guide*.

If your Elastic Beanstalk environment uses the .NET on Windows Server platform, see Getting Started with Amazon EC2 Windows Instances in the *Amazon EC2 User Guide*.

Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment

This topic explains the types of instance logs that Elastic Beanstalk provides. It also provides detailed instructions for retreiveing and managing them.

The Amazon EC2 instances in your Elastic Beanstalk environment generate logs that you can view to troubleshoot issues with your application or configuration files. Logs created by the web server, application server, Elastic Beanstalk platform scripts, and AWS CloudFormation are stored locally on individual instances. You can easily retrieve them by using the environment management console or the EB CLI. You can also configure your environment to stream logs to Amazon CloudWatch Logs in real time.

Tail logs are the last 100 lines of the most commonly used log files—Elastic Beanstalk operational logs and logs from the web server or application server. When you request tail logs in the environment management console or with **eb logs**, an instance in your environment concatenates the most recent log entries into a single text file and uploads it to Amazon S3.

Bundle logs are full logs for a wider range of log files, including logs from yum and cron and several logs from AWS CloudFormation. When you request bundle logs, an instance in your environment packages the full log files into a ZIP archive and uploads it to Amazon S3.

To upload rotated logs to Amazon S3, the instances in your environment must have an instance profile with permission to write to your Elastic Beanstalk Amazon S3 bucket. These permissions are included in the default instance profile that Elastic Beanstalk prompts you to create when you launch an environment in the Elastic Beanstalk console for the first time.

To retrieve instance logs

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

3. In the navigation pane, choose **Logs**.
4. Choose **Request Logs**, and then choose the type of logs to retrieve. To get tail logs, choose **Last 100 Lines**. To get bundle logs, choose **Full Logs**.
5. When Elastic Beanstalk finishes retrieving your logs, choose **Download**.

Elastic Beanstalk stores tail and bundle logs in an Amazon S3 bucket, and generates a presigned Amazon S3 URL that you can use to access your logs. Elastic Beanstalk deletes the files from Amazon S3 after a duration of 15 minutes.

Warning

Anyone in possession of the presigned Amazon S3 URL can access the files before they are deleted. Make the URL available only to trusted parties.

Note

Your user policy must have the `s3:DeleteObject` permission. Elastic Beanstalk uses your user permissions to delete the logs from Amazon S3.

To persist logs, you can configure your environment to publish logs to Amazon S3 automatically after they are rotated. To enable log rotation to Amazon S3, follow the procedure in Configuring instance log viewing. Instances in your environment will attempt to upload logs that have been rotated once per hour.

If your application generates logs in a location that isn't part of the default configuration for your environment's platform, you can extend the default configuration by using configuration files (`.ebextensions`). You can add your application's log files to tail logs, bundle logs, or log rotation.

For real-time log streaming and long-term storage, configure your environment to stream logs to Amazon CloudWatch Logs.

Sections

- Log location on Amazon EC2 instances
- Log location in Amazon S3
- Log rotation settings on Linux

- Extending the default log task configuration
- Streaming log files to Amazon CloudWatch Logs

Log location on Amazon EC2 instances

Logs are stored in standard locations on the Amazon EC2 instances in your environment. Elastic Beanstalk generates the following logs.

Amazon Linux 2

- `/var/log/eb-engine.log`

Amazon Linux AMI (AL1)

Note

On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information about migrating to a current and fully supported Amazon Linux 2023 platform branch, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

- `/var/log/eb-activity.log`
- `/var/log/eb-commandprocessor.log`

Windows Server

- `C:\Program Files\Amazon\ElasticBeanstalk\logs\`
- `C:\cfn\log\cfn-init.log`

These logs contain messages about deployment activities, including messages related to configuration files (`.ebextensions`).

Each application and web server stores logs in its own folder:

- **Apache** – `/var/log/httpd/`

- **IIS** – `C:\inetpub\wwwroot\`
- **Node.js** – `/var/log/nodejs/`
- **nginx** – `/var/log/nginx/`
- **Passenger** – `/var/app/support/logs/`
- **Puma** – `/var/log/puma/`
- **Python** – `/opt/python/log/`
- **Tomcat** – `/var/log/tomcat/`

Log location in Amazon S3

When you request tail or bundle logs from your environment, or when instances upload rotated logs, they're stored in your Elastic Beanstalk bucket in Amazon S3. Elastic Beanstalk creates a bucket named `elasticbeanstalk-`*`region`*`-`*`account-id`* for each AWS Region in which you create environments. Within this bucket, logs are stored under the path `resources/environments/logs/`*`logtype`*`/`*`environment-id`*`/`*`instance-id`*.

For example, logs from instance `i-0a1fd158`, in Elastic Beanstalk environment `e-mpcwnwheky` in AWS Region `us-west-2` in account `123456789012`, are stored in the following locations:

- **Tail Logs** –

  ```
  s3://elasticbeanstalk-us-west-2-123456789012/resources/environments/logs/tail/e-mpcwnwheky/i-0a1fd158
  ```
- **Bundle Logs** –

  ```
  s3://elasticbeanstalk-us-west-2-123456789012/resources/environments/logs/bundle/e-mpcwnwheky/i-0a1fd158
  ```
- **Rotated Logs** –

  ```
  s3://elasticbeanstalk-us-west-2-123456789012/resources/environments/logs/publish/e-mpcwnwheky/i-0a1fd158
  ```

Note

You can find your environment ID in the environment management console.

Elastic Beanstalk deletes tail and bundle logs from Amazon S3 automatically 15 minutes after they are created. Rotated logs persist until you delete them or move them to S3 Glacier.

Log rotation settings on Linux

On Linux platforms, Elastic Beanstalk uses `logrotate` to rotate logs periodically. If configured, after a log is rotated locally, the log rotation task picks it up and uploads it to Amazon S3. Logs that are rotated locally don't appear in tail or bundle logs by default.

You can find Elastic Beanstalk configuration files for `logrotate` in `/etc/logrotate.elasticbeanstalk.hourly/`. These rotation settings are specific to the platform, and might change in future versions of the platform. For more information about the available settings and example configurations, run `man logrotate`.

The configuration files are invoked by cron jobs in `/etc/cron.hourly/`. For more information about `cron`, run `man cron`.

Extending the default log task configuration

Elastic Beanstalk uses files in subfolders of `/opt/elasticbeanstalk/tasks` (Linux) or `C:\Program Files\Amazon\ElasticBeanstalk\config` (Windows Server) on the Amazon EC2 instance to configure tasks for tail logs, bundle logs, and log rotation.

On Amazon Linux:

- **Tail Logs –**

  ```
  /opt/elasticbeanstalk/tasks/taillogs.d/
  ```
- **Bundle Logs –**

  ```
  /opt/elasticbeanstalk/tasks/bundlelogs.d/
  ```
- **Rotated Logs –**

  ```
  /opt/elasticbeanstalk/tasks/publishlogs.d/
  ```

On Windows Server:

- **Tail Logs –**

  ```
  c:\Program Files\Amazon\ElasticBeanstalk\config\taillogs.d\
  ```

- **Bundle Logs –**

  `c:\Program Files\Amazon\ElasticBeanstalk\config\bundlelogs.d\`
- **Rotated Logs –**

  `c:\Program Files\Amazon\ElasticBeanstalk\config\publogs.d\`

For example, the `eb-activity.conf` file on Linux adds two log files to the tail logs task.

`/opt/elasticbeanstalk/tasks/taillogs.d/eb-activity.conf`

```
/var/log/eb-commandprocessor.log
/var/log/eb-activity.log
```

You can use environment configuration files (`.ebextensions`) to add your own `.conf` files to these folders. A `.conf` file lists log files specific to your application, which Elastic Beanstalk adds to the log file tasks.

Use the `files` section to add configuration files to the tasks that you want to modify. For example, the following configuration text adds a log configuration file to each instance in your environment. This log configuration file, `cloud-init.conf`, adds `/var/log/cloud-init.log` to tail logs.

```
files:
  "/opt/elasticbeanstalk/tasks/taillogs.d/cloud-init.conf" :
    mode: "000755"
    owner: root
    group: root
    content: |
      /var/log/cloud-init.log
```

Add this text to a file with the `.config` file name extension to your source bundle under a folder named `.ebextensions`.

```
~/workspace/my-app
|-- .ebextensions
|   `-- tail-logs.config
|-- index.php
`-- styles.css
```

On Linux platforms, you can also use wildcard characters in log task configurations. This configuration file adds all files with the `.log` file name extension from the `log` folder in the application root to bundle logs.

```
files:
  "/opt/elasticbeanstalk/tasks/bundlelogs.d/applogs.conf" :
    mode: "000755"
    owner: root
    group: root
    content: |
      /var/app/current/log/*.log
```

Log task configurations don't support wildcard characters on Windows platforms.

Note

To help familiarize yourself with log customization procedures, you can deploy a sample application using the EB CLI. For this, the EB CLI creates a local application directory that contains an `.ebextentions` subdirectory with a sample configuration. You can also use the sample application's log files to explore the log retrieval feature described in this topic.

For more information about using configuration files, see Advanced environment customization with configuration files (`.ebextensions`).

Much like extending tail logs and bundle logs, you can extend log rotation using a configuration file. Whenever Elastic Beanstalk rotates its own logs and uploads them to Amazon S3, it also rotates and uploads your additional logs. Log rotation extension behaves differently depending on the platform's operating system. The following sections describe the two cases.

Extending log rotation on Linux

As explained in Log rotation settings on Linux, Elastic Beanstalk uses `logrotate` to rotate logs on Linux platforms. When you configure your application's log files for log rotation, the application doesn't need to create copies of log files. Elastic Beanstalk configures `logrotate` to create a copy of your application's log files for each rotation. Therefore, the application must keep log files unlocked when it isn't actively writing to them.

Extending log rotation on Windows server

On Windows Server, when you configure your application's log files for log rotation, the application must rotate the log files periodically. Elastic Beanstalk looks for files with names starting with the pattern you configured, and picks them up for uploading to Amazon S3. In addition, periods in the file name are ignored, and Elastic Beanstalk considers the name up to the period to be the base log file name.

Elastic Beanstalk uploads all versions of a base log file except for the newest one, because it considers that one to be the active application log file, which can potentially be locked. Your application can, therefore, keep the active log file locked between rotations.

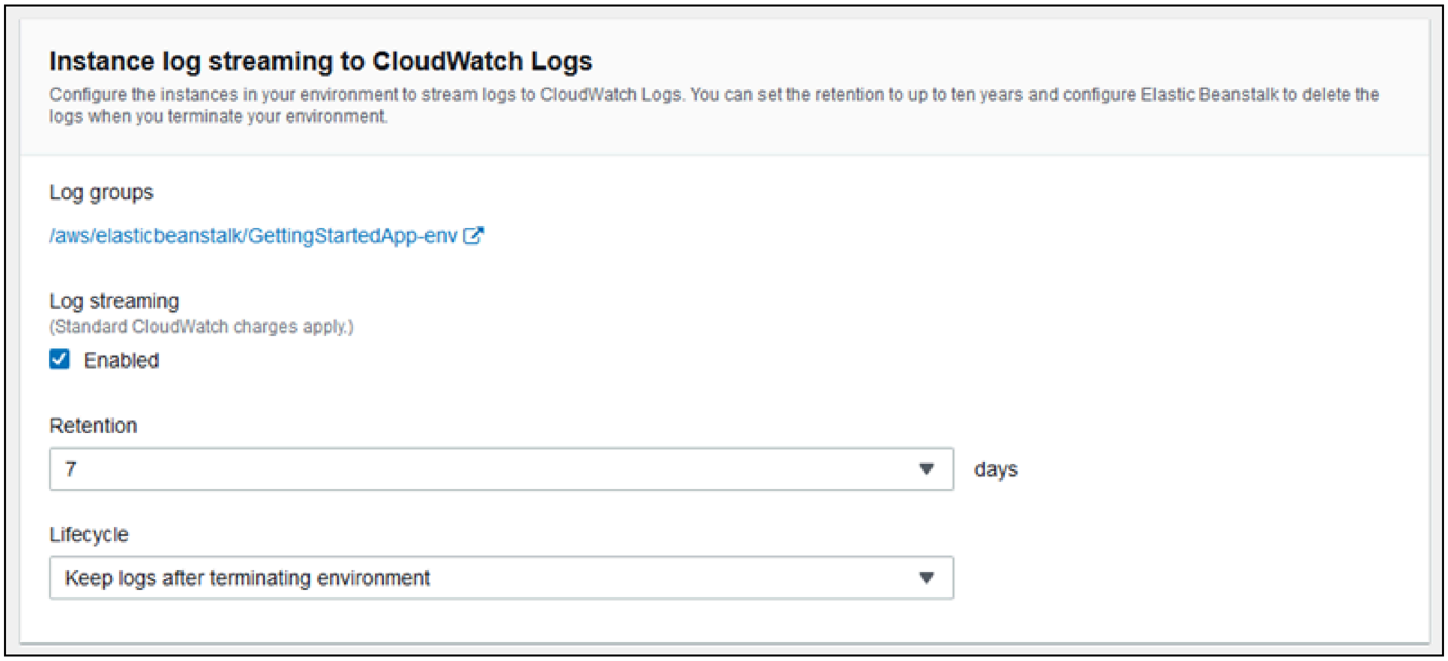
For example, your application writes to a log file named `my_log.log`, and you specify this name in your `.conf` file. The application periodically rotates the file. During the Elastic Beanstalk rotation cycle, it finds the following files in the log file's folder: `my_log.log`, `my_log.0800.log`, `my_log.0830.log`. Elastic Beanstalk considers all of them to be versions of the base name `my_log`. The file `my_log.log` has the latest modification time, so Elastic Beanstalk uploads only the other two files, `my_log.0800.log` and `my_log.0830.log`.

Streaming log files to Amazon CloudWatch Logs

You can configure your environment to stream logs to Amazon CloudWatch Logs in the Elastic Beanstalk console or by using configuration options. With CloudWatch Logs, each instance in your environment streams logs to log groups that you can configure to be retained for weeks or years, even after your environment is terminated.

The set of logs streamed varies per environment, but always includes `eb-engine.log` and access logs from the nginx or Apache proxy server that runs in front of your application.

You can configure log streaming in the Elastic Beanstalk console either during environment creation or for an existing environment. You can set the following options from the console: enable /disable log streaming to CloudWatch Logs, set the number of retention days, and select from Lifecyle options. In the following example, logs are saved for up to seven days, even when the environment is terminated.



Instance log streaming to CloudWatch Logs

Configure the instances in your environment to stream logs to CloudWatch Logs. You can set the retention to up to ten years and configure Elastic Beanstalk to delete the logs when you terminate your environment.

Log groups

/aws/elasticbeanstalk/GettingStartedApp-env

Log streaming
(Standard CloudWatch charges apply.)

☑ Enabled

Retention: 7 days

Lifecycle: Keep logs after terminating environment

The following configuration file enables log streaming with 180 days retention, even if the environment is terminated.

Example .ebextensions/log-streaming.config

```
option_settings:
  aws:elasticbeanstalk:cloudwatch:logs:
    StreamLogs: true
    DeleteOnTerminate: false
    RetentionInDays: 180
```

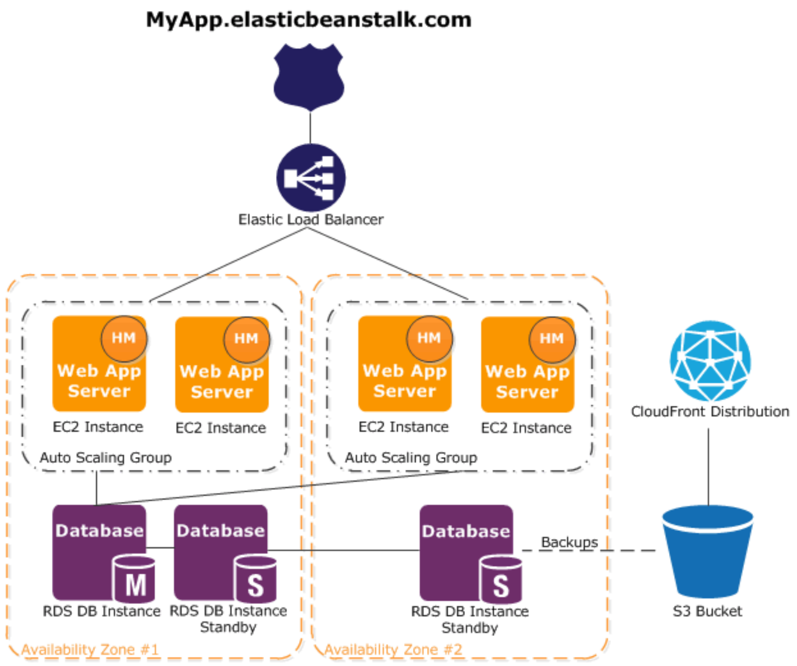
Using Elastic Beanstalk with other AWS services

The topics in this section describe the many ways you can use additional AWS services with your Elastic Beanstalk application. To implement your application's environments, Elastic Beanstalk manages resources of other AWS services or uses their functionality. In addition, Elastic Beanstalk integrates with AWS services that it doesn't use directly as part of your environments.

Topics

Architectural overview

The following diagram illustrates an example architecture of Elastic Beanstalk across multiple Availability Zones working with other AWS products such as Amazon CloudFront, Amazon Simple Storage Service (Amazon S3), and Amazon Relational Database Service (Amazon RDS).



To plan for fault-tolerance, it is advisable to have N+1 Amazon EC2 instances and spread your instances across multiple Availability Zones. In the unlikely case that one Availability Zone goes down, you will still have your other Amazon EC2 instances running in another Availability Zone. You can adjust Amazon EC2 Auto Scaling to allow for a minimum number of instances as well as multiple Availability Zones. For instructions on how to do this, see Auto Scaling your Elastic Beanstalk environment instances. For more information about building fault-tolerant applications, go to Building Fault-Tolerant Applications on AWS.

The following sections discuss in more detail integration with Amazon CloudFront, Amazon CloudWatch, Amazon DynamoDB Amazon ElastiCache, Amazon RDS, Amazon Route 53, Amazon Simple Storage Service, Amazon VPC , and IAM.

Using Elastic Beanstalk with Amazon CloudFront

Amazon CloudFront is a web service that speeds up distribution of your static and dynamic web content, for example, .html, .css, .php, image, and media files, to end users. CloudFront delivers your content through a worldwide network of edge locations. When an end user requests content that you're serving with CloudFront, the user is routed to the edge location that provides the lowest latency, so content is delivered with the best possible performance. If the content is already in that edge location, CloudFront delivers it immediately. If the content is not currently in that edge location, CloudFront retrieves it from an Amazon S3 bucket or an HTTP server (for example, a web server) that you have identified as the source for the definitive version of your content.

After you have created and deployed your Elastic Beanstalk application you can sign up for CloudFront and start using CloudFront to distribute your content. Learn more about CloudFront from the Amazon CloudFront Developer Guide.

Logging Elastic Beanstalk API calls with AWS CloudTrail

Elastic Beanstalk is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Elastic Beanstalk. CloudTrail captures all API calls for Elastic Beanstalk as events, including calls from the Elastic Beanstalk console, from the EB CLI, and from your code to the Elastic Beanstalk APIs. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Elastic Beanstalk. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Elastic Beanstalk, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Elastic Beanstalk information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Elastic Beanstalk, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Elastic Beanstalk, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all regions. The trail logs events from all regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All Elastic Beanstalk actions are logged by CloudTrail and are documented in the AWS Elastic Beanstalk API Reference. For example, calls to the `DescribeApplications`, `UpdateEnvironment`, and `ListTagsForResource` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understanding Elastic Beanstalk log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `UpdateEnvironment` action called by an IAM user named `intern`, for the `sample-env` environment in the `sample-app` application.

```
{
  "Records": [{
    "eventVersion": "1.05",
    "userIdentity": {
      "type": "IAMUser",
      "principalId": "AIXDAYQEXAMPLEUMLYNGL",
      "arn": "arn:aws:iam::123456789012:user/intern",
      "accountId": "123456789012",
      "accessKeyId": "ASXIAGXEXAMPLEQULKNXV",
      "userName": "intern",
      "sessionContext": {
        "attributes": {
          "mfaAuthenticated": "false",
          "creationDate": "2016-04-22T00:23:24Z"
        }
      },
      "invokedBy": "signin.amazonaws.com"
    },
    "eventTime": "2016-04-22T00:24:14Z",
    "eventSource": "elasticbeanstalk.amazonaws.com",
    "eventName": "UpdateEnvironment",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "255.255.255.54",
    "userAgent": "signin.amazonaws.com",
    "requestParameters": {
      "applicationName": "sample-app",
      "environmentName": "sample-env",
      "optionSettings": []
    },
    "responseElements": null,
    "requestID": "84ae9ecf-0280-17ce-8612-705c7b132321",
    "eventID": "e48b6a08-c6be-4a22-99e1-c53139cbfb18",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
  }]
}
```

Using Elastic Beanstalk with Amazon CloudWatch

Amazon CloudWatch enables you to monitor, manage, and publish various metrics, as well as configure alarm actions based on data from metrics. Amazon CloudWatch monitoring enables you to collect, analyze, and view system and application metrics so that you can make operational and business decisions more quickly and with greater confidence.

You can use Amazon CloudWatch to collect metrics about your Amazon Web Services (AWS) resources—such as the performance of your Amazon EC2 instances. You can also publish your own metrics directly to Amazon CloudWatch. Amazon CloudWatch alarms help you implement decisions more easily by enabling you to send notifications or automatically make changes to the resources you are monitoring, based on rules that you define. For example, you can create alarms that initiate Amazon EC2 Auto Scaling and Amazon Simple Notification Service (Amazon SNS) actions on your behalf.

Elastic Beanstalk automatically uses Amazon CloudWatch to help you monitor your application and environment status. You can navigate to the Amazon CloudWatch console to see your dashboard and get an overview of all of your resources as well as your alarms. You can also choose to view more metrics or add custom metrics.

For more information about Amazon CloudWatch, go to the Amazon CloudWatch Developer Guide. For an example of how to use Amazon CloudWatch with Elastic Beanstalk, see the section called "Example: Using custom Amazon CloudWatch metrics".

Using Elastic Beanstalk with Amazon CloudWatch Logs

This topic explains the monitoring features that the Amazon CloudWatch Logs service can provide to Elastic Beanstalk. It also walks you through the configuration setup and lists the locations of the logs for each Elastic Beanstalk platform.

Implementing CloudWatch Logs can enable you to do the following monitoring activities:

- Monitor and archive your Elastic Beanstalk application, system, and custom log files from the Amazon EC2 instances of your environments.
- Configure alarms that make it easier for you to react to specific log stream events that your metric filters extract.

The CloudWatch Logs agent installed on each Amazon EC2 instance in your environment publishes metric data points to the CloudWatch service for each log group you configure. Each log group applies its own filter patterns to determine what log stream events to send to CloudWatch as data points. Log streams that belong to the same log group share the same retention, monitoring, and access control settings. You can configure Elastic Beanstalk to automatically stream logs to the CloudWatch service, as described in Streaming instance logs to CloudWatch Logs. For more information about CloudWatch Logs, including terminology and concepts, see the Amazon CloudWatch Logs User Guide.

In addition to instance logs, if you enable enhanced health for your environment, you can configure the environment to stream health information to CloudWatch Logs. See Streaming Elastic Beanstalk environment health information to Amazon CloudWatch Logs.

Topics

- Troubleshooting CloudWatch Logs integration
- Streaming Elastic Beanstalk environment health information to Amazon CloudWatch Logs

Prerequisites to instance log streaming to CloudWatch Logs

To enable streaming of logs from your environment's Amazon EC2 instances to CloudWatch Logs, you must meet the following conditions.

- *Platform* – Because this feature is only available in platform versions released on or after this release, if you are using an earlier platform version, update your environment to a current one.
- If you don't have the *AWSElasticBeanstalkWebTier* or *AWSElasticBeanstalkWorkerTier* Elastic Beanstalk managed policy in your Elastic Beanstalk instance profile, you must add the following to your profile to enable this feature.

  JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "logs:PutLogEvents",
      "logs:CreateLogStream"
    ],
    "Resource": [
    "*"
    ]
  }
  ]
}
```

How Elastic Beanstalk sets up CloudWatch Logs

Elastic Beanstalk installs a CloudWatch log agent with the default configuration settings on each instance it creates. Learn more in the CloudWatch Logs Agent Reference.

When you enable instance log streaming to CloudWatch Logs, Elastic Beanstalk sends log files from your environment's instances to CloudWatch Logs. Different platforms stream different logs. The following table lists the logs, by platform.

Platform / Platform Branch	Logs
Docker / Platform Branch: Docker Running on 64bit Amazon Linux 2	• /var/log/eb-engine.log • /var/log/eb-hooks.log • /var/log/docker • /var/log/docker-events.log • /var/log/eb-docker/containers/eb-current-app/stdouterr.log • /var/log/nginx/access.log • /var/log/nginx/error.log
Docker / Platform Branch: ECS Running on 64bit Amazon Linux 2	• /var/log/docker-events.log • /var/log/eb-ecs-mgr.log • /var/log/eb-engine.log • /var/log/eb-hooks.log • /var/log/ecs/ecs-agent.log • /var/log/ecs/ecs-init.log
Go .NET Core on Linux Java / Platform Branch: Corretto running on 64bit Amazon Linux 2	• /var/log/eb-engine.log • /var/log/eb-hooks.log • /var/log/web.stdout.log • /var/log/nginx/access.log • /var/log/nginx/error.log
Node.js Python	• /var/log/eb-engine.log • /var/log/eb-hooks.log • /var/log/web.stdout.log • /var/log/httpd/access_log • /var/log/httpd/error_log • /var/log/nginx/access.log

Platform / Platform Branch	Logs
	• /var/log/nginx/error.log
Tomcat PHP	• /var/log/eb-engine.log • /var/log/eb-hooks.log • /var/log/httpd/access_log • /var/log/httpd/error_log • /var/log/nginx/access.log • /var/log/nginx/error.log
.NET on Windows Server	• C:\inetpub\logs\LogFiles\W3SVC1\u_ex*.log • C:\Program Files\Amazon\ElasticBeanstalk\logs\AWSDeployment.log • C:\Program Files\Amazon\ElasticBeanstalk\logs\Hooks.log
Ruby	• /var/log/eb-engine.log • /var/log/eb-hooks.log • /var/log/puma/puma.log • /var/log/web.stdout.log • /var/log/nginx/access.log • /var/log/nginx/error.log

Log files on Amazon Linux AMI platforms

Note

On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. For more information about migrating to a current and fully supported Amazon Linux 2023 platform branch, see Migrating your Elastic Beanstalk Linux application to Amazon Linux 2023 or Amazon Linux 2.

The following table lists the log files streamed from instances on platform branches based on Amazon Linux AMI (preceding Amazon Linux 2), by platform.

Platform / Platform Branch	Logs
Docker / Platform Branch: Docker Running on 64bit Amazon Linux	• /var/log/eb-activity.log • /var/log/nginx/error.log • /var/log/docker-events.log • /var/log/docker • /var/log/nginx/access.log • /var/log/eb-docker/containers/eb-current-app/stdouterr.log
Docker / Platform Branch: Multicont ainer Docker Running on 64bit Amazon Linux	• /var/log/eb-activity.log • /var/log/ecs/ecs-init.log • /var/log/eb-ecs-mgr.log • /var/log/ecs/ecs-agent.log • /var/log/docker-events.log
Glassfish (Preconfigured Docker)	• /var/log/eb-activity.log • /var/log/nginx/error.log • /var/log/docker-events.log • /var/log/docker • /var/log/nginx/access.log
Go	• /var/log/eb-activity.log • /var/log/nginx/error.log • /var/log/nginx/access.log
Java / Platform Branch: Java 8 running on 64bit Amazon Linux Platform Branch: Java 7 running on 64bit Amazon Linux	• /var/log/eb-activity.log • /var/log/nginx/access.log • /var/log/nginx/error.log • /var/log/web-1.error.log • /var/log/web-1.log

Platform / Platform Branch	Logs
Tomcat	• /var/log/eb-activity.log • /var/log/httpd/error_log • /var/log/httpd/access_log • /var/log/nginx/error_log • /var/log/nginx/access_log
Node.js	• /var/log/eb-activity.log • /var/log/nodejs/nodejs.log • /var/log/nginx/error.log • /var/log/nginx/access.log • /var/log/httpd/error.log • /var/log/httpd/access.log
PHP	• /var/log/eb-activity.log • /var/log/httpd/error_log • /var/log/httpd/access_log
Python	• /var/log/eb-activity.log • /var/log/httpd/error_log • /var/log/httpd/access_log • /opt/python/log/supervisord.log
Ruby / Platform Branch: Puma with Ruby running on 64bit Amazon Linux	• /var/log/eb-activity.log • /var/log/nginx/error.log • /var/log/puma/puma.log • /var/log/nginx/access.log
Ruby / Platform Branch: Passenger with Ruby running on 64bit Amazon Linux	• /var/log/eb-activity.log • /var/app/support/logs/passenger.log • /var/app/support/logs/access.log • /var/app/support/logs/error.log

Elastic Beanstalk configures log groups in CloudWatch Logs for the various log files that it streams. To retrieve specific log files from CloudWatch Logs, you have to know the name of the corresponding log group. The log group naming scheme depends on the platform's operating system.

For Linux platforms, prefix the on-instance log file location with `/aws/elasticbeanstalk/`*`environment_name`* to get the log group name. For example, to retrieve the file `/var/log/nginx/error.log`, specify the log group `/aws/elasticbeanstalk/`*`environment_name`*`/var/log/nginx/error.log`.

For Windows platforms, see the following table for the log group corresponding to each log file.

On-instance log file	Log group
`C:\Program Files\Amazon\ElasticBeanstalk\logs\AWSDeployment.log`	`/aws/elasticbeanstalk/<environment-name>/EBDeploy-Log`
`C:\Program Files\Amazon\ElasticBeanstalk\logs\Hooks.log`	`/aws/elasticbeanstalk/<environment-name>/EBHooks-Log`
`C:\inetpub\logs\LogFiles` (the entire directory)	`/aws/elasticbeanstalk/<environment-name>/IIS-Log`

Streaming instance logs to CloudWatch Logs

You can enable instance log streaming to CloudWatch Logs using the Elastic Beanstalk console, the EB CLI, or configuration options.

Before you enable it, set up IAM permissions to use with the CloudWatch Logs agent. You can attach the following custom policy to the instance profile that you assign to your environment.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Instance log streaming using the Elastic Beanstalk console

To stream instance logs to CloudWatch Logs

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. Under **Instance log streaming to CloudWatch Logs**:
   - Enable **Log streaming**.
   - Set **Retention** to the number of days to save the logs.
   - Select the **Lifecycle** setting that determines whether the logs are saved after the environment is terminated.
6. To save the changes choose **Apply** at the bottom of the page.

After you enable log streaming, you can return to the **Software** configuration category or page and find the **Log Groups** link. Click this link to see your logs in the CloudWatch console.

Instance log streaming using the EB CLI

To enable instance log streaming to CloudWatch Logs using the EB CLI, use the **eb logs** command.

```
$ eb logs --cloudwatch-logs enable
```

You can also use **eb logs** to retrieve logs from CloudWatch Logs. You can retrieve all the environment's instance logs, or use the command's many options to specify subsets of logs to retrieve. For example, the following command retrieves the complete set of instance logs for your environment, and saves them to a directory under `.elasticbeanstalk/logs`.

```
$ eb logs --all
```

In particular, the `--log-group` option enables you to retrieve instance logs of a specific log group, corresponding to a specific on-instance log file. To do that, you need to know the name of the log group that corresponds to the log file you want to retrieve. You can find this information in How Elastic Beanstalk sets up CloudWatch Logs.

Instance log streaming using configuration files

When you create or update an environment, you can use a configuration file to set up and configure instance log streaming to CloudWatch Logs. The following example configuration file enables default instance log streaming. Elastic Beanstalk streams the default set of log files for your environment's platform. To use the example, copy the text into a file with the `.config` extension in the `.ebextensions` directory at the top level of your application source bundle.

```
option_settings:
  - namespace: aws:elasticbeanstalk:cloudwatch:logs
    option_name: StreamLogs
    value: true
```

Custom log file streaming

The Elastic Beanstalk integration with CloudWatch Logs doesn't directly support the streaming of custom log files that your application generates. To stream custom logs, use a configuration file to directly install the CloudWatch agent and to configure the files to be pushed. For an example configuration file, see `logs-streamtocloudwatch-linux.config`.

Note

The example doesn't work on the Windows platform.

For more information about configuring CloudWatch Logs, see the CloudWatch agent configuration file reference in the *Amazon CloudWatch User Guide*.

Troubleshooting CloudWatch Logs integration

Unable to locate environment instance logs

If you can't find some of the environment's instance logs that you expect in CloudWatch Logs, investigate the following common issues:

- Your IAM role lacks the required IAM permissions.
- You launched your environment in an AWS Region that doesn't support CloudWatch Logs.
- One of your custom log files doesn't exist in the path you specified.

Application logs missing or intermittent

If your Elastic Beanstalk application logs, (`/var/log/web.stdout.log`), appear to be missing or intermittent, this may be due to default rate-limiting settings in rsyslog and journald. While disabling rate-limiting entirely can resolve this issue, it's not recommended as it could lead to excessive disk usage, potential denial of service, or system performance degradation during unexpected log bursts. Instead, you can adjust the rate limits using the following `.ebextensions configuration`. This configuration increases the rate limit interval to 600 seconds with higher burst limits, providing a balance between proper logging and system protection.

Throttling issues

If an Elastic Beanstalk operation that concurrently launches a large number of instances returns a message like `Error: fail to create log stream: ThrottlingException: Rate exceeded`, it's throttling from too many calls to the CloudWatch API.

To resolve the throttling issue take one of the following actions:

- Use a smaller batch size with rolling deployments to reduce concurrent updates.
- Request an increase for your AWS account's Transaction Per Second (TPS) limit service quota for *CreateLogStream*. For more information, see CloudWatch Logs quotas and Managing your CloudWatch Logs service quotas in the *Amazon CloudWatch Logs User Guide*.

Streaming Elastic Beanstalk environment health information to Amazon CloudWatch Logs

If you enable enhanced health reporting for your environment, you can configure the environment to stream health information to CloudWatch Logs. This streaming is independent from Amazon EC2 instance log streaming. This topic describes environment health information streaming. For information about instance log streaming, see Using Elastic Beanstalk with Amazon CloudWatch Logs.

When you configure environment health streaming, Elastic Beanstalk creates a CloudWatch Logs log group for environment health. The log group's name is `/aws/elasticbeanstalk/`*`environment-name`*`/environment-health.log`. Within this log group, Elastic Beanstalk creates log streams named *`YYYY-MM-DD`*`#`*`<hash-suffix>`* (there might be more than one log stream per date).

When the environment's health status changes, Elastic Beanstalk adds a record to the health log stream. The record represents the health status transition—the new status and a description of the cause of change. For example, an environment's status might change to Severe because the load balancer is failing. For a description of enhanced health statuses, see Health colors and statuses.

Prerequisites to environment health streaming to CloudWatch Logs

To enable environment health streaming to CloudWatch Logs, you must meet the following conditions:

- *Platform* – You must be using a platform version that supports enhanced health reporting.
- *Permissions* – You must grant certain logging-related permissions to Elastic Beanstalk so that it can act on your behalf to stream health information for your environment. If your environment isn't using a service role that Elastic Beanstalk created for it, `aws-elasticbeanstalk-service-role`, or your account's service-linked role, `AWSServiceRoleForElasticBeanstalk`, be sure to add the following permissions to your custom service role.

```
{
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogStreams",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:log-group:/aws/elasticbeanstalk/*:log-stream:*"
}
```

Streaming environment health logs to CloudWatch Logs

You can enable environment health streaming to CloudWatch Logs using the Elastic Beanstalk console, the EB CLI, or configuration options.

Environment health log streaming using the Elastic Beanstalk console

To stream environment health logs to CloudWatch Logs

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Monitoring** configuration category, choose **Edit**.
5. Under **Health reporting**, make sure that the reporting **System** is set to **Enhanced**.
6. Under **Health event streaming to CloudWatch Logs**
    - Enable **Log streaming**.
    - Set **Retention** to the number of days to save the logs.
    - Select the **Lifecycle** setting that determines whether the logs are saved after the environment is terminated.
7. To save the changes choose **Apply** at the bottom of the page.

After you enable log streaming, you can return to the **Monitoring** configuration category or page and find the **Log Group** link. Click this link to see your environment health logs in the CloudWatch console.

Environment health log streaming using the EB CLI

To enable environment health log streaming to CloudWatch Logs using the EB CLI, use the **eb logs** command.

```
$ eb logs --cloudwatch-logs enable --cloudwatch-log-source environment-health
```

You can also use **eb logs** to retrieve logs from CloudWatch Logs. For example, the following command retrieves all the health logs for your environment, and saves them to a directory under `.elasticbeanstalk/logs`.

```
$ eb logs --all --cloudwatch-log-source environment-health
```

Environment health log streaming using configuration files

When you create or update an environment, you can use a configuration file to set up and configure environment health log streaming to CloudWatch Logs. To use the example below, copy the text into a file with the `.config` extension in the `.ebextensions` directory at the top level of your application source bundle. The example configures Elastic Beanstalk to enable environment health log streaming, keep the logs after terminating the environment, and save them for 30 days.

Example Health streaming configuration file

```
############################################################################
##  Sets up Elastic Beanstalk to stream environment health information
##  to Amazon CloudWatch Logs.
##  Works only for environments that have enhanced health reporting enabled.
############################################################################

option_settings:
  aws:elasticbeanstalk:cloudwatch:logs:health:
    HealthStreamingEnabled: true
    ### Settings below this line are optional.
    # DeleteOnTerminate: Delete the log group when the environment is
    # terminated. Default is false. If false, the health data is kept
    # RetentionInDays days.
    DeleteOnTerminate: false
    # RetentionInDays: The number of days to keep the archived health data
    # before it expires, if DeleteOnTerminate isn't set. Default is 7 days.
    RetentionInDays: 30
```

For option defaults and valid values, see `aws:elasticbeanstalk:cloudwatch:logs:health`.

Using Elastic Beanstalk with Amazon EventBridge

Using Amazon EventBridge, you can set up event-driven rules that monitor your Elastic Beanstalk resources and initiate target actions that use other AWS services. For example, you can set a rule for sending out email notifications by signaling an Amazon SNS topic whenever the health of a production environment changes to a *Warning* status. Or, you can set a Lambda function to pass a notification to Slack whenever the health of your environment changes to a *Degraded* or *Severe* status.

You can create rules in Amazon EventBridge to act on any of the following Elastic Beanstalk events:

- *State changes for environment operations (including create, update, and terminate operations).* The event specifies if the state change has started, succeeded, or failed.
- *State changes for other resources.* Besides environments, other resources that are monitored include load balancers, auto scaling groups, and instances.
- *Health transition for environments.* The event states where the environment health has transitioned from one health status to another one.
- *State change for managed updates.* The event specifies if the state change has started, succeeded, or failed.

To capture specific Elastic Beanstalk events that you're interested in, define event-specific patterns that EventBridge can use to detect the events. Event patterns have the same structure as the events they match. The pattern quotes the fields that you want to match and provides the values that you're looking for. Events are emitted on a best effort basis. They're delivered from Elastic Beanstalk to EventBridge in near real-time under normal operational circumstances. However, situations can arise that may delay or prevent delivery of an event.

For a list of fields that are contained in Elastic Beanstalk events and their possible string values, see Elastic Beanstalk event field mapping. For information about how EventBridge rules work with event patterns, see Events and Event Patterns in EventBridge.

Monitor an Elastic Beanstalk resource with EventBridge

With EventBridge, you can create rules that define actions to take when Elastic Beanstalk emits events for its resources. For example, you can create a rule that sends you an email message whenever the status of an environment changes.

The EventBridge console has a **Pre-defined pattern** option for building Elastic Beanstalk event patterns. If you select this option in the EventBridge console when you create a rule, you can build an Elastic Beanstalk event pattern quickly. You only need to select the event fields and values. As you make selections, the console builds and displays the event pattern. Alternatively, you can manually edit the event pattern that you build and can save it as a custom pattern. The console also provides you the option to display a detailed **Sample Event** that you can copy and paste to the event pattern that you're building.

If you prefer to type or copy and paste an event pattern into the EventBridge console, you can select to use the **Custom pattern** option in the console. By doing this, you don't need to go through the steps of selecting fields and values described earlier. This topic offers examples of both event-matching patterns and Elastic Beanstalk events that you can use.

To create a rule for a resource event

1. Log in to AWS using an account that has permissions to use EventBridge and Elastic Beanstalk.
2. Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
3. In the navigation pane, choose **Rules**.
4. Choose **Create rule**.
5. Enter a **Name** for the rule, and, optionally, a description.
6. For **Event bus**, choose **default**. When an AWS service in your account emits an event, it always goes to your account's default event bus.
7. For **Rule type**, choose **Rule with an event pattern**.
8. Choose **Next**.
9. For **Event source**, choose **AWS events or EventBridge partner events**.
10. (Optional) For **Sample event**, select **AWS events**. Enter *Elastic Beanstalk* in the search field. This will provide a list of sample Elastic Beanstalk events from which you can choose to display. This step simply displays a sample event that you can reference. It doesn't affect the outcome of the rule creation. The Example Elastic Beanstalk events section later in this topic provides examples of the same type of events.
11. In the **Event pattern** section, choose **Event pattern form**.

    Note

    If you already have text for an event pattern and don't need the EventBridge console to build it for you, select **Custom pattern (JSON editor)**. You can then either manually enter or copy and paste text into the **Event pattern** box. Choose **Next**, and go to the step about entering a target.

12. For **Event source**, choose **AWS services**.
13. For **AWS service**, select **Elastic Beanstalk**.

14. For **Event type**, select **Status Change**.
15. This step covers how you can work with the **detail type**, **status**, and **severity** event fields for Elastic Beanstalk. As you choose these fields and the values you want to match, the console builds and displays the event pattern.

    - If you select *only one* value for **Specific detail type(s)**, you can choose one or more values for the next field in the hierarchy.
    - If you choose *more than one* value for **Specific detail type(s)**, do not choose specific values for the next fields in the hierarchy. This prevents ambiguous matching logic across fields in your event pattern.

    The **environment** event field isn't affected by this hierarchy, so it displays as described in the next step.
16. For environment, select **Any environment** or **Specific environment(s)**.

    - If you select **Specific environment(s)**, you can choose one or more environments from the dropdown list. EventBridge adds all of the environments that you select inside the *EnvironmentName[ ]* list in the *detail* section of the event pattern. Then, your rule filters all events to include only the specific environments that you choose.
    - If you select **Any environment**, then no environments are added to your event pattern. Because of this, your rule doesn't filter any of the Elastic Beanstalk events based on environment.
17. Choose **Next**.
18. For **Target types**, choose **AWS service**.
19. For **Select a target**, choose the target action to take when a resource state change event is received from Elastic Beanstalk.

    For example, you can use an Amazon Simple Notification Service (SNS) topic to send an email or text message when an event occurs. To do this, you need to create an Amazon SNS topic using the Amazon SNS console. To learn more, see Using Amazon SNS for user notifications.

    Important

    Some target actions might require the use of other services and incur additional charges, such as the Amazon SNS or Lambda service. For more information about AWS pricing, see https://aws.amazon.com/pricing/. Some services are part of the AWS Free

Usage Tier. If you are a new customer, you can test drive these services for free. See https://aws.amazon.com/free/ for more information.

20. (Optional) Choose **Add another target** to specify an additional target action for the event rule.
21. Choose **Next**.
22. (Optional) Enter one or more tags for the rule. For more information, see Amazon EventBridge tags in the *Amazon EventBridge User Guide*.
23. Choose **Next**.
24. Review the details of the rule and choose **Create rule**.

Example Elastic Beanstalk event patterns

Event patterns have the same structure as the events they match. The pattern quotes the fields that you want to match and provides the values that you're looking for.

- *Health status change* for all environments

```
{
   "source": [
    "aws.elasticbeanstalk"
  ],
  "detail-type": [
    "Health status change"
    ]
}
```

- *Health status change* for the following environments: `myEnvironment1` and `myEnvironment2`. This event pattern filters for these two specific environments, whereas the previous *Health status change* example that doesn't filter sends events for all environments.

```
{"source": [
    "aws.elasticbeanstalk"
    ],
    "detail-type": [
        "Health status change"
    ],
    "detail": {
        "EnvironmentName": [
            "myEnvironment1",
            "myEnvironment2"
        ]
    }
}
```

- *Elastic Beanstalk resource status change* for all environments

```
{
  "source": [
    "aws.elasticbeanstalk"
  ],
  "detail-type": [
    "Elastic Beanstalk resource status change"
    ]
}
```

- *Elastic Beanstalk resource status change* with `Status` *Environment update failed* and `Severity` *ERROR* for the following environments: `myEnvironment1` and `myEnvironment2`

```
{"source": [
    "aws.elasticbeanstalk"
    ],
    "detail-type": [
        "Elastic Beanstalk resource status change"
    ],
    "detail": {
        "Status": [
            "Environment update failed"
            ],
        "Severity": [
            "ERROR"
            ],
        "EnvironmentName": [
            "myEnvironment1",
            "myEnvironment2"
        ]
    }
}
```

- *Other resource status change* for load balancers, auto scaling groups, and instances

```
{
   "source": [
     "aws.elasticbeanstalk"
   ],
   "detail-type": [
     "Other resource status change"
     ]
}
```

- *Managed update status change* for all environments

```
{
     "source": [
      "aws.elasticbeanstalk"
    ],
    "detail-type": [
      "Managed update status change"
      ]
}
```

- To capture *all events* from Elastic Beanstalk (exclude the `detail-type` section)

```
{
    "source": [
      "aws.elasticbeanstalk"
    ]
}
```

Example Elastic Beanstalk events

The following is an example Elastic Beanstalk event for a *resource status change*:

```
{
   "version":"0",
   "id":"1234a678-1b23-c123-12fd3f456e78",
   "detail-type":"Elastic Beanstalk resource status change",
   "source":"aws.elasticbeanstalk",
   "account":"111122223333",
   "time":"2020-11-03T00:31:54Z",
   "region":"us-east-1",
   "resources":[
      "arn:was:elasticbeanstalk:us-east-1:111122223333:environment/myApplication/
myEnvironment"
   ],
   "detail":{
      "Status":"Environment creation started",
      "EventDate":1604363513951,
      "ApplicationName":"myApplication",
      "Message":"createEnvironment is starting.",
      "EnvironmentName":"myEnvironment",
      "Severity":"INFO"
   }
}
```

The following is an example Elastic Beanstalk event for a *health status change*:

```
{
   "version":"0",
   "id":"1234a678-1b23-c123-12fd3f456e78",
   "detail-type":"Health status change",
   "source":"aws.elasticbeanstalk",
   "account":"111122223333",
   "time":"2020-11-03T00:34:48Z",
   "region":"us-east-1",
   "resources":[
      "arn:was:elasticbeanstalk:us-east-1:111122223333:environment/myApplication/
myEnvironment"
   ],
   "detail":{
      "Status":"Environment health changed",
      "EventDate":1604363687870,
      "ApplicationName":"myApplication",
      "Message":"Environment health has transitioned from Pending to Ok. Initialization
 completed 1 second ago and took 2 minutes.",
      "EnvironmentName":"myEnvironment",
      "Severity":"INFO"
   }
}
```

Elastic Beanstalk event field mapping

The following table maps Elastic Beanstalk event fields and their possible string values to the EventBridge `detail-type` field. For more information about how EventBridge works with event patterns for a service, see Events and Event Patterns in EventBridge.

EventBridge field *detail-type*	Elastic Beanstalk field *Status*	Elastic Beanstalk field *Severity*	Elastic Beanstalk field *Message*
Elastic Beanstalk resource status change	Environment creation started	INFO	createEnvironment is starting.
	Environment creation successful	INFO	createEnvironment completed successfully.
	Environment creation successful	INFO	Launched environment: <Environment Name>. However, there were issues during launch. See event log for details.
	Environment creation failed	ERROR	Failed to launch environment.
	Environment update started	INFO	Environment update is starting.
	Environment update successful	INFO	Environment update completed successfully.
	Environment update failed	ERROR	Failed to deploy configuration.
	Environment termination started	INFO	terminateEnvironment is starting.

EventBridge field *detail-type*	**Elastic Beanstalk field *Status***	**Elastic Beanstalk field *Severity***	**Elastic Beanstalk field *Message***
	Environment termination successful	INFO	terminateEnvironment completed successfully.
	Environment termination failed	INFO	The environment termination step failed because at least one of the environment termination workflows failed.
Other resource status change	Auto Scaling group created	INFO	createEnvironment is starting.
	Auto Scaling group deleted	INFO	createEnvironment is starting.
	Instance added	INFO	Added instance [i-123456789a12b1234] to your environment.
	Instance removed	INFO	Removed instance [i-123456789a12b1234] from your environment.
	Load balancer created	INFO	Created load balancer named: <LB Name>

EventBridge field *detail-type*	Elastic Beanstalk field *Status*	Elastic Beanstalk field *Severity*	Elastic Beanstalk field *Message*
	Load balancer deleted	INFO	Deleted load balancer named: <LB Name>
Health status change	Environment health changed	INFO/ WARN	Environment health has transitioned to <healthStatus>.
	Environment health changed	INFO/ WARN	Environment health has transitioned from <healthStatus> to <healthStatus>.
Managed update status change	Managed updated started	INFO	Managed platform update is in-progress.
	Managed update failed	INFO	Managed update failed, retrying in %s minutes.

Finding and tracking Elastic Beanstalk resources with AWS Config

AWS Config provides a detailed view of the configuration of AWS resources in your AWS account. You can see how resources are related, get a history of configuration changes, and see how relationships and configurations change over time. You can use AWS Config to define rules that evaluate resource configurations for data compliance.

Several Elastic Beanstalk resource types are integrated with AWS Config:

- Applications
- Application Versions

- Environments

The following section shows how to configure AWS Config to record resources of these types.

For more information about AWS Config, see the AWS Config Developer Guide. For pricing information, see the AWS Config pricing information page.

Setting up AWS Config

To initially set up AWS Config, see the following topics in the AWS Config Developer Guide.

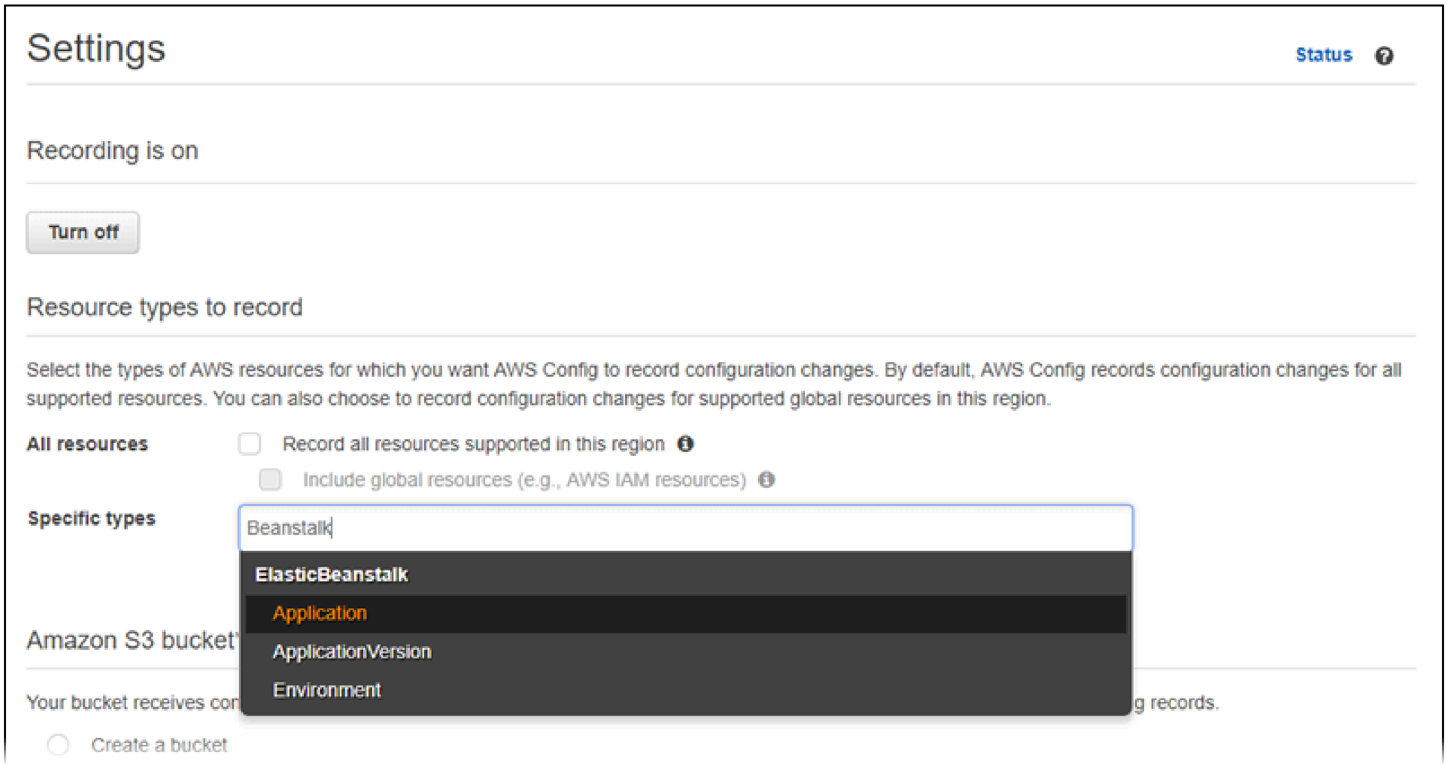
- Setting up AWS Config with the Console
- Setting up AWS Config with the AWS CLI

Configuring AWS Config to record Elastic Beanstalk resources

By default, AWS Config records configuration changes for all supported types of *regional resources* that it discovers in the region in which your environment is running. You can customize AWS Config to record changes only for specific resource types, or changes to *global resources*.

For example, you can configure AWS Config to record changes for Elastic Beanstalk resources and a subset of other AWS resources that Elastic Beanstalk starts for you. Using the AWS Config Console, you can select Elastic Beanstalk as a resource in the AWS Config **Settings** page from the **Specific Types** field. From there you can choose to record any of the Elastic Beanstalk resource types: **Application**, **ApplicationVersion**, and **Environment**.

The following figure shows the AWS Config **Settings** page, with Elastic Beanstalk resource types that you can choose to record: **Application**, **ApplicationVersion**, and **Environment**.


Settings

Status

Recording is on

Turn off

Resource types to record

Select the types of AWS resources for which you want AWS Config to record configuration changes. By default, AWS Config records configuration changes for all supported resources. You can also choose to record configuration changes for supported global resources in this region.

All resources: Record all resources supported in this region

Include global resources (e.g., AWS IAM resources)

Specific types: Beanstalk

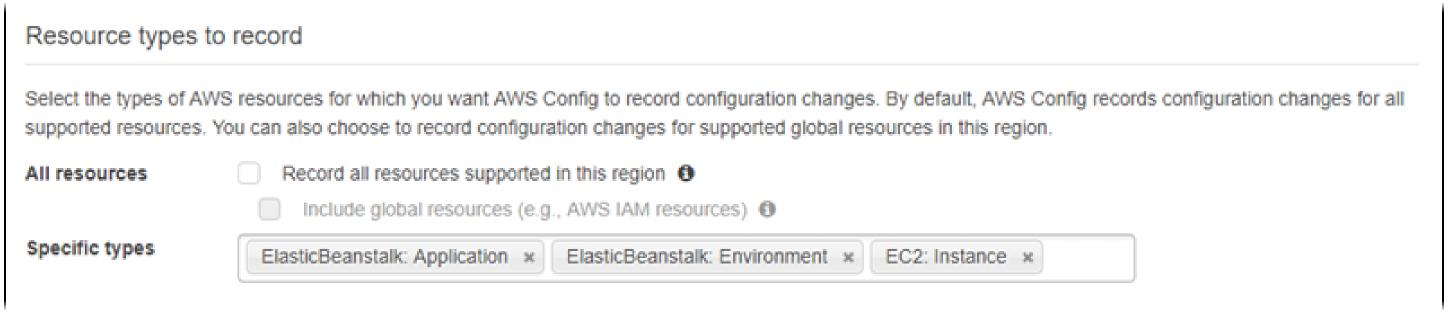
ElasticBeanstalk
- Application
- ApplicationVersion
- Environment

Amazon S3 bucket

Your bucket receives co… g records.

Create a bucket

After you select a few resource types, this is how the **Specific types** list appears.


Resource types to record

Select the types of AWS resources for which you want AWS Config to record configuration changes. By default, AWS Config records configuration changes for all supported resources. You can also choose to record configuration changes for supported global resources in this region.

All resources: Record all resources supported in this region

Include global resources (e.g., AWS IAM resources)

Specific types: ElasticBeanstalk: Application × ElasticBeanstalk: Environment × EC2: Instance ×

To learn about *regional* vs. *global* resources, and for the full customization procedure, see Selecting which Resources AWS Config Records.

Viewing Elastic Beanstalk configuration details in the AWS Config console

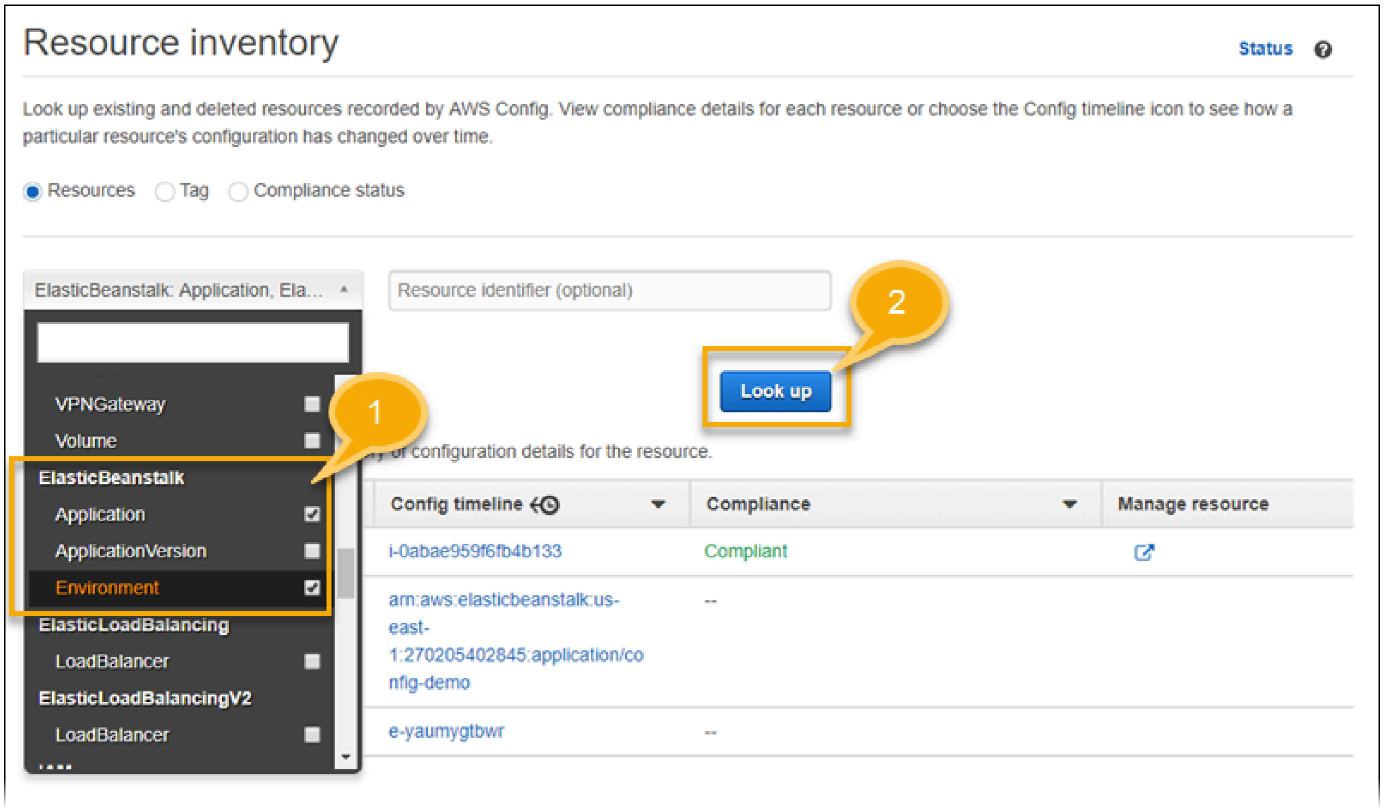
You can use the AWS Config console to look for Elastic Beanstalk resources, and get current and historical details about their configurations. The following example shows how to find information about an Elastic Beanstalk environment.

To find an Elastic Beanstalk environment in the AWS Config console

1. Open the [AWS Config console](https://console.aws.amazon.com/config/).
2. Choose **Resources**.
3. On the **Resource inventory** page, choose **Resources**.
4. Open the **Resource type** menu, scroll to **ElasticBeanstalk**, and then choose one or more of the Elastic Beanstalk resource types.

   > **Note**
   >
   > To view configuration details for other resources that Elastic Beanstalk created for your application, choose additional resource types. For example, you can choose **Instance** under **EC2**.

5. Choose **Look up**. See **2** in the following figure.

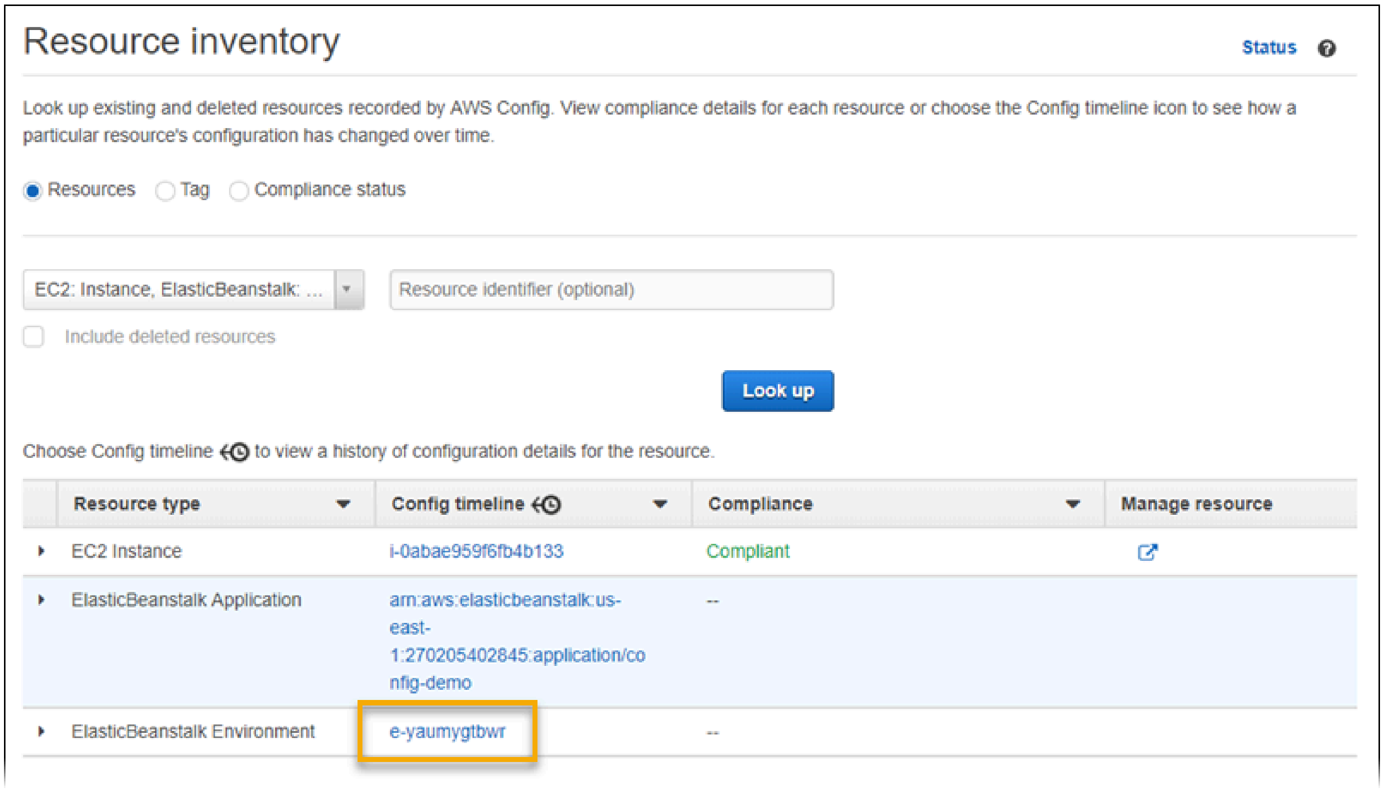


Resource inventory

Look up existing and deleted resources recorded by AWS Config. View compliance details for each resource or choose the Config timeline icon to see how a particular resource's configuration has changed over time.

Config timeline	Compliance	Manage resource
i-0abae959f6fb4b133	Compliant	
arn:aws:elasticbeanstalk:us-east-1:270205402845:application/config-demo	--	
e-yaumygtbwr	--	

6. Choose a resource ID in the list of resources that AWS Config displays.



Resource inventory

Look up existing and deleted resources recorded by AWS Config. View compliance details for each resource or choose the Config timeline icon to see how a particular resource's configuration has changed over time.

[✓ Resources] Tag Compliance status

EC2: Instance, ElasticBeanstalk: ...

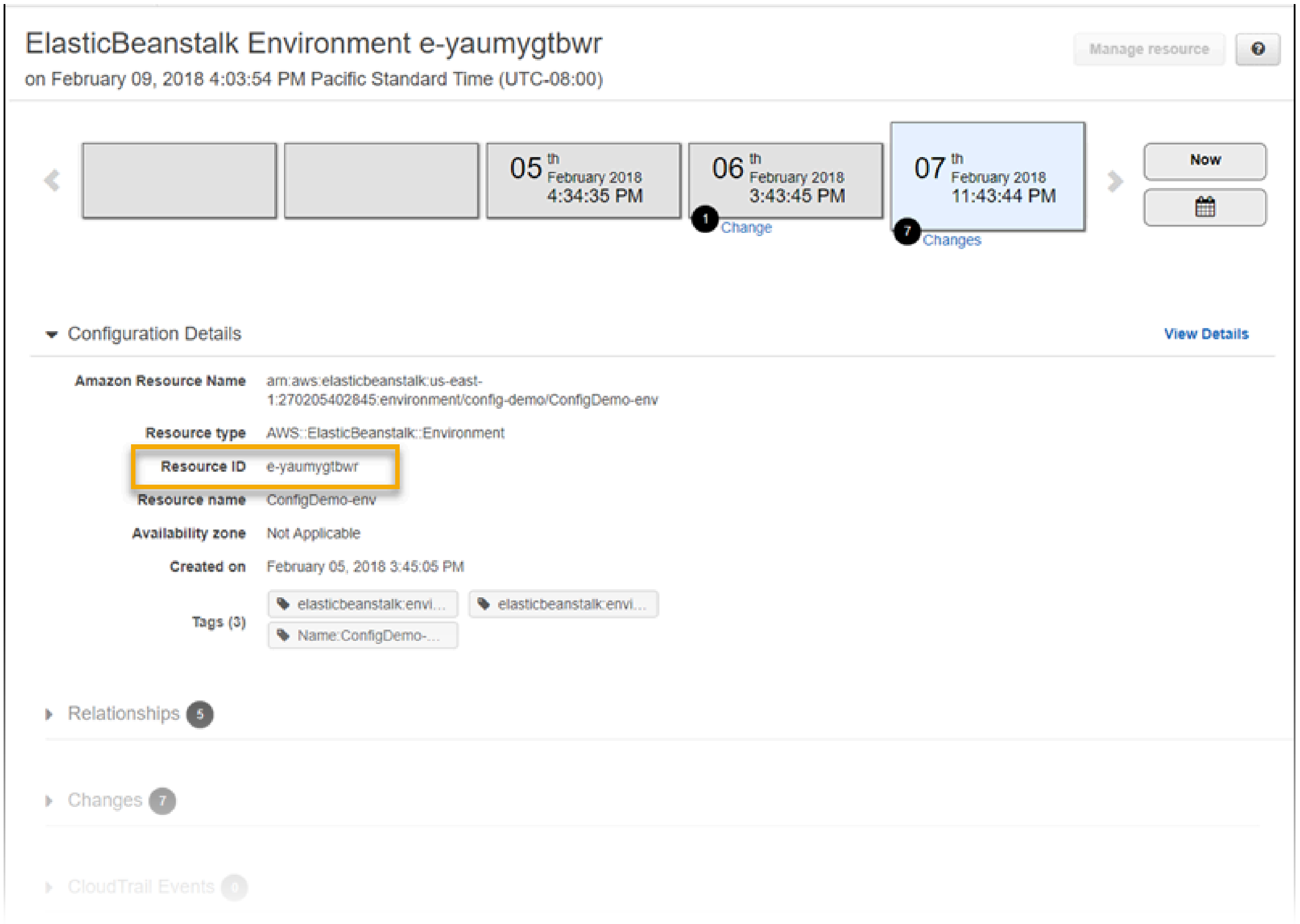
Resource identifier (optional)

Include deleted resources

Choose Config timeline to view a history of configuration details for the resource.

Resource type	Config timeline	Compliance	Manage resource
EC2 Instance	i-0abae959f6fb4b133	Compliant	
ElasticBeanstalk Application	arn:aws:elasticbeanstalk:us-east-1:270205402845:application/config-demo	--	
ElasticBeanstalk Environment	e-yaumygtbwr	--	

AWS Config displays configuration details and other information about the resource you selected.



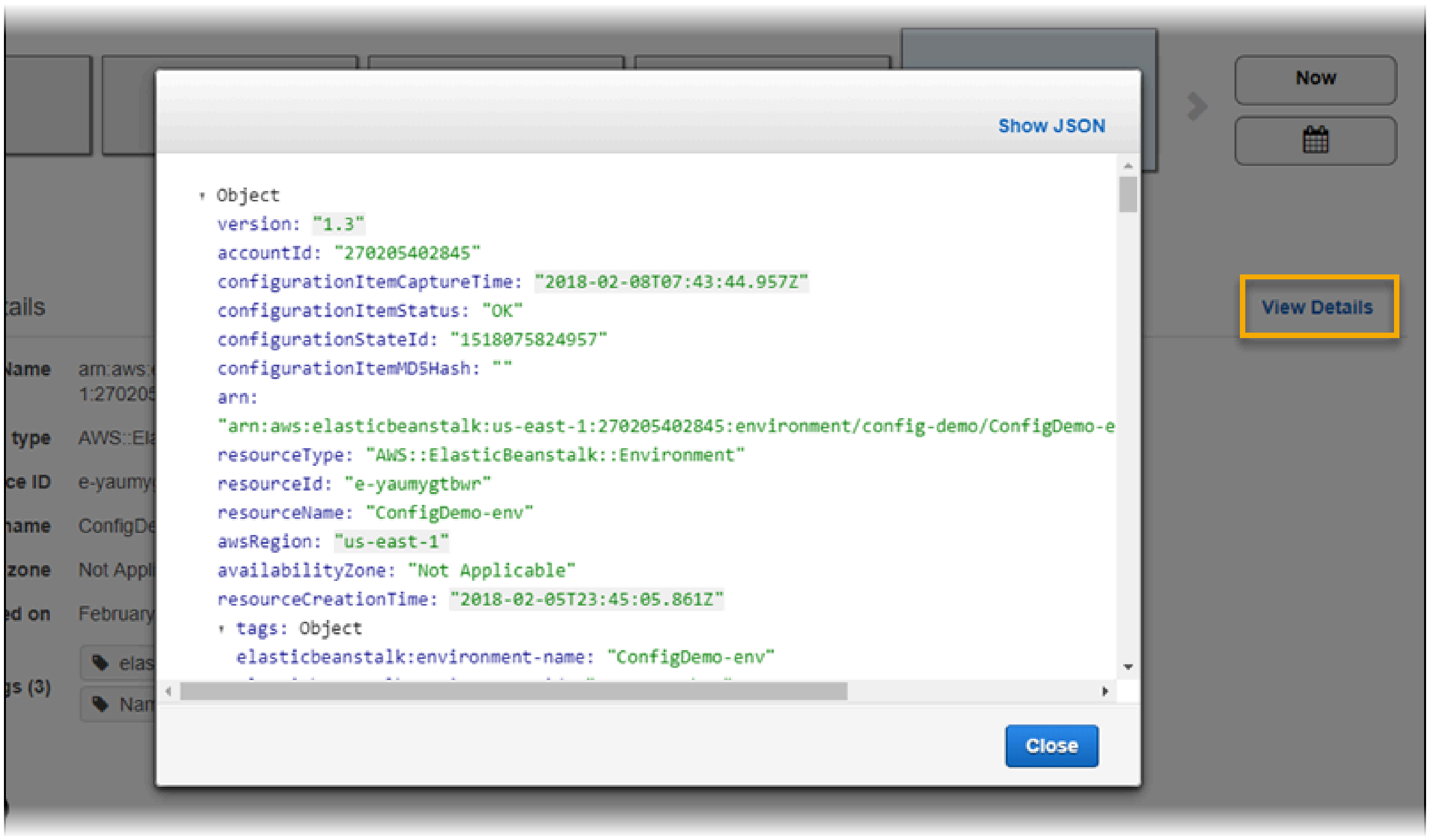
ElasticBeanstalk Environment e-yaumygtbwr
on February 09, 2018 4:03:54 PM Pacific Standard Time (UTC-08:00)

Manage resource
05th February 2018 4:34:35 PM
06th February 2018 3:43:45 PM
1 Change
07th February 2018 11:43:44 PM
7 Changes
Now

Configuration Details — View Details

Amazon Resource Name	arn:aws:elasticbeanstalk:us-east-1:270205402845:environment/config-demo/ConfigDemo-env
Resource type	AWS::ElasticBeanstalk::Environment
Resource ID	e-yaumygtbwr
Resource name	ConfigDemo-env
Availability zone	Not Applicable
Created on	February 05, 2018 3:45:05 PM
Tags (3)	elasticbeanstalk:envi... elasticbeanstalk:envi... Name:ConfigDemo-...

To see the full details of the recorded configuration, choose **View Details**.



```
▾ Object
  version: "1.3"
  accountId: "270205402845"
  configurationItemCaptureTime: "2018-02-08T07:43:44.957Z"
  configurationItemStatus: "OK"
  configurationStateId: "1518075824957"
  configurationItemMD5Hash: ""
  arn:
  "arn:aws:elasticbeanstalk:us-east-1:270205402845:environment/config-demo/ConfigDemo-e
  resourceType: "AWS::ElasticBeanstalk::Environment"
  resourceId: "e-yaumygtbwr"
  resourceName: "ConfigDemo-env"
  awsRegion: "us-east-1"
  availabilityZone: "Not Applicable"
  resourceCreationTime: "2018-02-05T23:45:05.861Z"
  ▾ tags: Object
    elasticbeanstalk:environment-name: "ConfigDemo-env"
```

To learn more ways to find a resource and view information on this page, see Viewing AWS Resource Configurations and History in the *AWS Config Developer Guide*.

Evaluating Elastic Beanstalk resources using AWS Config rules

You can create AWS Config rules, which represent the ideal configuration settings for your Elastic Beanstalk resources. You can use predefined *AWS Managed Config Rules*, or define custom rules. AWS Config continuously tracks changes to the configuration of your resources to determine whether those changes violate any of the conditions in your rules. The AWS Config console shows the compliance status of your rules and resources.

If a resource violates a rule and is flagged as *noncompliant*, AWS Config can alert you using an Amazon Simple Notification Service (Amazon SNS) topic. To programmatically consume the data in these AWS Config alerts, use an Amazon Simple Queue Service (Amazon SQS) queue as the notification endpoint for the Amazon SNS topic. For example, you might want to write code that starts a workflow when someone modifies your environment's Auto Scaling group configuration.

To learn more about setting up and using rules, see Evaluating Resources with AWS Config Rules in the *AWS Config Developer Guide*.

Using Elastic Beanstalk with Amazon DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. If you are a developer, you can use DynamoDB to create a database table that can store and retrieve any amount of data, and serve any level of request traffic. DynamoDB automatically spreads the data and traffic for the table over a sufficient number of servers to handle the request capacity specified by the customer and the amount of data stored, while maintaining consistent and fast performance. All data items are stored on solid state drives (SSDs) and are automatically replicated across multiple Availability Zones in an AWS Region to provide built-in high availability and data durability.

If you use periodic tasks in a worker environment, Elastic Beanstalk creates a DynamoDB table and uses it to perform leader election and store information about the task. Each instance in the environment attempts to write to the table every few seconds to become leader and perform the task when scheduled.

You can use configuration files to create a DynamoDB table for your application. See eb-node-express-sample on GitHub for a sample Node.js application that creates a table with a configuration file and connects to it with the AWS SDK for JavaScript in Node.js. For an example walkthrough using DynamoDB with PHP, see Example: DynamoDB, CloudWatch, and SNS. For an example that uses the AWS SDK for Java, see Manage Tomcat Session State with DynamoDB in the AWS SDK for Java documentation.

When you create a DynamoDB table using configuration files, the table isn't tied to your environment's lifecycle, and isn't deleted when you terminate your environment. To ensure that personal information isn't unnecessarily retained, delete any records that you don't need anymore, or delete the table.

For more information about DynamoDB, see the DynamoDB Developer Guide.

Using Elastic Beanstalk with Amazon ElastiCache

Amazon ElastiCache is a web service that enables setting up, managing, and scaling distributed in-memory cache environments in the cloud. It provides a high-performance, scalable, and cost-effective in-memory cache, while removing the complexity associated with deploying and managing a distributed cache environment. ElastiCache is protocol-compliant with Redis and Memcached, so the code, applications, and most popular tools that you use today with your existing Redis and Memcached environments will work seamlessly with the service. For more information about ElastiCache, go to the Amazon ElastiCache product page.

To use Elastic Beanstalk with Amazon ElastiCache

1. Create an ElastiCache cluster.
   - For instructions on how to create an ElastiCache cluster with Redis, go to Getting Started with Amazon ElastiCache for Redis in the *ElastiCache for Redis User Guide*.
   - For instructions on how to create an ElastiCache cluster with Memcached, go to Getting Started with Amazon ElastiCache for Memcached in the *ElastiCache for Memcached User Guide*.
2. Configure your ElastiCache Security Group to allow access from the Amazon EC2 security group used by your Elastic Beanstalk application. For instructions on how to find the name of your EC2 security group using the AWS Management Console, see EC2 security groups on the *EC2 Instances* document page.
   - For more information on Redis, go to Authorize Access in the *ElastiCache for Redis User Guide*.
   - For more information on Memcached, go to Authorize Access in the *ElastiCache for Memcached User Guide*.

You can use configuration files to customize your Elastic Beanstalk environment to use ElastiCache. For configuration file examples that integrate ElastiCache with Elastic Beanstalk, see Example: ElastiCache.

Using Elastic Beanstalk with Amazon Elastic File System

With Amazon Elastic File System (Amazon EFS), you can create network file systems that can be mounted by instances across multiple Availability Zones. An Amazon EFS file system is an AWS resource that uses security groups to control access over the network that's in your default or custom VPC.

In an Elastic Beanstalk environment, you can use Amazon EFS to create a shared directory that stores files for your application that users upload and modify. Your application can treat a mounted Amazon EFS volume such as local storage. That way, you don't have to change your application code to scale up to multiple instances.

For more information about Amazon EFS, see the Amazon Elastic File System User Guide.

Note

Elastic Beanstalk creates a *webapp* user that you can set up as the owner for application directories on Amazon EC2 instances. For more information, see Persistent Storage in the *Design considerations* topic of this guide.

Sections

Configuration files

Elastic Beanstalk provides configuration files that you can use to create and mount Amazon EFS file systems. You can create an Amazon EFS volume as part of your environment, or mount an Amazon EFS volume that you created independently of Elastic Beanstalk.

- **storage-efs-createfilesystem.config** – Uses the `Resources` key to create a new file system and mount points in Amazon EFS. All instances in your environment can connect to the same file system for shared, scalable storage. Use `storage-efs-mountfilesystem.config` to mount the file system on each instance.

  Internal resources

  Any resources that you create with configuration files are tied to the lifecycle of your environment. If you terminate your environment or remove the configuration file, these resources are lost.

- **storage-efs-mountfilesystem.config** – Mount an Amazon EFS file system to a local path on the instances in your environment. You can create the volume as part of the environment with `storage-efs-createfilesystem.config`. Or, you can mount it to your environment using the Amazon EFS console, AWS CLI, or AWS SDK.

To use the configuration files, start by creating your Amazon EFS file system with `storage-efs-createfilesystem.config`. Follow the instructions in the configuration file and add it to the .ebextensions directory in your source code to create the file system in your VPC.

Deploy your updated source code to your Elastic Beanstalk environment. This is to confirm that the file system was created successfully. Then, add the `storage-efs-mountfilesystem.config` to mount the file system to the instances in your environment. Doing this in two separate deployments ensures that, if the mount operation fails, the file system is kept intact. If you do both in the same deployment, an issue with either step will cause the file system to terminate when the deployment fails.

Encrypted file systems

Amazon EFS supports encrypted file systems. The `storage-efs-createfilesystem.config` configuration file that's discussed in this topic defines two custom options. You can use these options to create an Amazon EFS encrypted file system. For more information, refer to the instructions in the configuration file.

Sample applications

Elastic Beanstalk also provides sample applications that use Amazon EFS for shared storage. The two projects have configuration files that you can use with a standard WordPress or Drupal installer to run a blog or other content management system in a load-balanced environment. When a user uploads a photo or other media, the file is stored on an Amazon EFS file system. This avoids having to use the alternative, which is using a plugin to store uploaded files in Amazon S3.

- **Load-balanced WordPress** – This includes the configuration files to install WordPress securely and run it in a load-balanced Elastic Beanstalk environment.
- **Load-balanced Drupal** – This includes the configuration files and instructions for installing Drupal securely and running it in a load-balanced Elastic Beanstalk environment.

Cleaning up file systems

If you created an Amazon EFS file system that uses a configuration file as part of your Elastic Beanstalk environment, Elastic Beanstalk removes the file system when you terminate the environment. To minimize storage costs of a running application, routinely delete files that your application doesn't need. Or, ensure that the application code maintains file lifecycle correctly.

Important

If you created an Amazon EFS file system that's outside of an Elastic Beanstalk environment and mounted it to the environment's instances, Elastic Beanstalk doesn't remove the file system when you terminate the environment. To ensure that your personal information isn't retained and avoid storage costs, delete the files that your application stored if you don't need them anymore. Alternatively, you can remove the entire file system.

Using Elastic Beanstalk with AWS Identity and Access Management

AWS Identity and Access Management (IAM) helps you securely control access to your AWS resources. This section includes reference materials for working with IAM policies, instance profiles, and service roles.

For an overview of permissions, see Elastic Beanstalk Service roles, instance profiles, and user policies. For most environments, the service role and instance profile that the Elastic Beanstalk console prompts you to create when you launch your first environment have all of the permissions that you need. Likewise, the managed policies provided by Elastic Beanstalk for full access and read-only access contain all of the user permissions required for daily use.

The IAM User Guide provides in-depth coverage of AWS permissions.

Topics

Managing Elastic Beanstalk instance profiles

An instance profile is a container for an AWS Identity and Access Management (IAM) role that you can use to pass role information to an Amazon EC2 instance when the instance starts.

If your AWS account doesn't have an EC2 instance profile, you must create one using the IAM service. You can then assign the EC2 instance profile to new environments that you create. The **Create environment** steps in the Elastic Beanstalk console provides you access to the IAM console, so that you can create an EC2 instance profile with the required permissions.

> **Note**
>
> Previously Elastic Beanstalk created a default EC2 instance profile named `aws-elasticbeanstalk-ec2-role` the first time an AWS account created an environment. This instance profile included default managed policies. If your account already has this instance profile, it will remain available for you to assign to your environments.
> However, recent AWS security guidelines don't allow an AWS service to automatically create roles with trust policies to other AWS services, EC2 in this case. Because of these security guidelines, Elastic Beanstalk no longer creates a default `aws-elasticbeanstalk-ec2-role` instance profile.

Managed policies

Elastic Beanstalk provides several managed policies to allow your environment to meet different use cases. To meet the default use cases for an environment, these policies must be attached to the role for the EC2 instance profile.

- **AWSElasticBeanstalkWebTier** – Grants permissions for the application to upload logs to Amazon S3 and debugging information to AWS X-Ray. To view the managed policy content, see AWSElasticBeanstalkWebTier in the *AWS Managed Policy Reference Guide*.
- **AWSElasticBeanstalkWorkerTier** – Grants permissions for log uploads, debugging, metric publication, and worker instance tasks, including queue management, leader election, and periodic tasks. To view the managed policy content, see AWSElasticBeanstalkWorkerTier in the *AWS Managed Policy Reference Guide*.
- **AWSElasticBeanstalkMulticontainerDocker** – Grants permissions for the Amazon Elastic Container Service to coordinate cluster tasks for Docker environments. To view the managed

policy content, see AWSElasticBeanstalkMulticontainerDocker in the *AWS Managed Policy Reference Guide*.

Important

Elastic Beanstalk managed policies don't provide granular permissions—they grant all permissions that are potentially needed for working with Elastic Beanstalk applications. In some cases you may wish to restrict the permissions of our managed policies further. For an example of one use case, see Preventing cross-environment Amazon S3 bucket access. Our managed policies also don't cover permissions to custom resources that you might add to your solution, and that aren't managed by Elastic Beanstalk. To implement more granular permissions, minimum required permissions, or custom resource permissions, use custom policies.

Trust relationship policy for EC2

To allow the EC2 instances in your environment to assume the required role, the instance profile must specify Amazon EC2 as a trusted entity in the trust relationship policy, as follows.

JSON

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

To customize permissions, you can add policies to the role attached to the default instance profile or create your own instance profile with a restricted set of permissions.

Sections

Creating an instance profile

An instance profile is a wrapper around a standard IAM role that allows an EC2 instance to assume the role. You can create an instance profile with the default Elastic Beanstalk managed policies. You can also create additional instance profiles to customize permissions for different applications. Or you can create an instance profile that doesn't include the two managed policies that grant permissions for worker tier or ECS managed Docker environments, if you don't use those features.

To create an instance profile with the default managed policies

1. Open the **Roles** page in the IAM console.
2. Choose **Create role**.
3. For **Trusted entity type**, choose **AWS service**.
4. For **Service or use case**, choose **Elastic Beanstalk**.
5. For **Use case**, choose **Elastic Beanstalk – Compute**.
6. Choose **Next**.
7. Enter a **Role name**.

   You can enter the name of the default role that the Elastic Beanstalk console suggests: `aws-elasticbeanstalk-ec2-role`.
8. Verify that **Permissions policies** include the following, then choose **Next**:
   - `AWSElasticBeanstalkWebTier`
   - `AWSElasticBeanstalkWorkerTier`
   - `AWSElasticBeanstalkMulticontainerDocker`
9. Choose **Create role**.

To create an instance profile with your specific choice of managed policies

1. Open the **Roles** page in the IAM console.
2. Choose **Create role**.
3. Under **Trusted entity type**, choose **AWS service**.
4. Under **Use case**, choose **EC2**.
5. Choose **Next**.
6. Attach the appropriate managed policies provided by Elastic Beanstalk and any additional policies that provide permissions that your application needs.
7. Choose **Next**.
8. Enter a name for the role.
9. (Optional) Add tags to the role.
10. Choose **Create role**.

Adding permissions to the default instance profile

If your application accesses AWS APIs or resources to which permissions aren't granted in the default instance profile, add policies that grant permissions in the IAM console.

To add policies to the role attached to the default instance profile

1. Open the Roles page in the IAM console.
2. Choose the role assigned as your EC2 instance profile.
3. On the **Permissions** tab, choose **Attach policies**.
4. Select the managed policy for the additional services that your application uses. For example, `AmazonS3FullAccess` or `AmazonDynamoDBFullAccess`.
5. Choose **Attach policy**.

Verifying the permissions assigned your instance profile

The permissions assigned to your default instance profile can vary depending on when it was created, the last time you launched an environment, and which client you used. You can verify the permissions on the default instance profile in the IAM console.

To verify the default instance profile's permissions

1. Open the **Roles** page in the IAM console.
2. Choose the role assigned as your EC2 instance profile.
3. On the **Permissions** tab, review the list of policies attached to the role.
4. To see the permissions that a policy grants, choose the policy.

Updating an out-of-date default instance profile

If the default instance profile lacks the required permissions, you can add the managed policies to the role assigned as your EC2 instance profile manually.

To add managed policies to the role attached to the default instance profile

1. Open the **Roles** page in the IAM console.
2. Choose the role assigned as your EC2 instance profile.
3. On the **Permissions** tab, choose **Attach policies**.
4. Type **`AWSElasticBeanstalk`** to filter the policies.
5. Select the following policies, and then choose **Attach policy**:
   - `AWSElasticBeanstalkWebTier`
   - `AWSElasticBeanstalkWorkerTier`
   - `AWSElasticBeanstalkMulticontainerDocker`

Managing Elastic Beanstalk service roles

To manage and monitor your environment, AWS Elastic Beanstalk performs actions on environment resources on your behalf. Elastic Beanstalk needs certain permissions to perform these actions, and it assumes AWS Identity and Access Management (IAM) service roles to get these permissions.

Elastic Beanstalk needs to use temporary security credentials whenever it assumes a service role. To get these credentials, Elastic Beanstalk sends a request to AWS Security Token Service (AWS STS) on a Region specific endpoint. For more information, see Temporary Security Credentials in the *IAM User Guide*.

Note

If the AWS STS endpoint for the Region where your environment is located is deactivated, Elastic Beanstalk sends the request on an alternative endpoint that can't be deactivated. This endpoint is associated with a different Region. Therefore, the request is a cross-Region request. For more information, see Activating and Deactivating AWS STS in an AWS Region in the *IAM User Guide*.

Managing service roles using the Elastic Beanstalk console and EB CLI

You can use the Elastic Beanstalk console and EB CLI to set up service roles for your environment with a sufficient set of permissions. They create a default service role and use managed policies in it.

Managed service role policies

Elastic Beanstalk provides one managed policy for enhanced health monitoring, and another one with additional permissions required for managed platform updates. The console and EB CLI assign both of these policies to the default service role that they create for you. These policies should only be used for this default service role. They should not be used with other users or roles in your accounts.

`AWSElasticBeanstalkEnhancedHealth`

This policy grants permissions for Elastic Beanstalk to monitor instance and environment health. It also includes Amazon SQS actions to allow Elastic Beanstalk to monitor queue activity for worker environments. To view the content of this managed policy, see the AWSElasticBeanstalkEnhancedHealth page in the *AWS Managed Policy Reference Guide*.

`AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy`

This policy grants permissions for Elastic Beanstalk to update environments on your behalf to perform managed platform updates. To view the content of this managed policy, see the AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy page in the *AWS Managed Policy Reference Guide*.

Service-level permission groupings

This policy is grouped into statements based on the set of permissions provided.

- *ElasticBeanstalkPermissions* – This group of permissions is for calling the Elastic Beanstalk service actions (Elastic Beanstalk APIs).
- *AllowPassRoleToElasticBeanstalkAndDownstreamServices* – This group of permissions allows any role to be passed to Elastic Beanstalk and to other downstream services like AWS CloudFormation.
- *ReadOnlyPermissions* – This group of permissions is for collecting information about the running environment.
- **OperationPermissions* – Groups with this naming pattern are for calling the necessary operations to perform platform updates.
- **BroadOperationPermissions* – Groups with this naming pattern are for calling the necessary operations to perform platform updates. They also include broad permissions for supporting legacy environments.
- **TagResource* – Groups with this naming pattern are for calls that use the tag-on-create APIs to attach tags on resources that are being created in an Elastic Beanstalk environment.

To view the content of a managed policy, you can also use the **Policies** page in the IAM console.

Important

Elastic Beanstalk managed policies don't provide granular permissions—they grant all permissions that are potentially needed for working with Elastic Beanstalk applications. In some cases you may wish to restrict the permissions of our managed policies further. For an example of one use case, see Preventing cross-environment Amazon S3 bucket access. Our managed policies also don't cover permissions to custom resources that you might add to your solution, and that aren't managed by Elastic Beanstalk. To implement more granular permissions, minimum required permissions, or custom resource permissions, use custom policies.

Deprecated managed policies

In the past, Elastic Beanstalk supported the **AWSElasticBeanstalkService** managed service role policy. This policy has been replaced by **AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy**. You might still be able to see and use the earlier policy in the IAM console.

To view the managed policy content, see AWSElasticBeanstalkService in the *AWS Managed Policy Reference Guide*.

However, we recommend that you transition to using the new managed policy (**AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy**). Add custom policies to grant permissions to custom resources, if you have any.

Using the Elastic Beanstalk console

When you launch an environment in the Elastic Beanstalk console, the console creates a default service role that's named `aws-elasticbeanstalk-service-role`, and attaches managed policies with default permissions to this service role.

To allow Elastic Beanstalk to assume the `aws-elasticbeanstalk-service-role` role, the service role specifies Elastic Beanstalk as a trusted entity in the trust relationship policy.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Sid": "",
        "Effect": "Allow",
        "Principal": {
          "Service": "elasticbeanstalk.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
          "StringEquals": {
            "sts:ExternalId": "elasticbeanstalk"
          }
        }
      }
    ]
}
```

When you enable managed platform updates for your environment, Elastic Beanstalk assumes a separate managed-updates service role to perform managed updates. By default, the Elastic Beanstalk console uses the same generated service role, `aws-elasticbeanstalk-service-`

`role`, for the managed-updates service role. If you change your default service role, the console sets the managed-updates service role to use the managed-updates service-linked role, `AWSServiceRoleForElasticBeanstalkManagedUpdates`. For more information about service-linked roles, see the section called "Using service-linked roles".

Note

Because of permission issues, the Elastic Beanstalk service doesn't always successfully create this service-linked role for you. Therefore, the console tries to explicitly create it. To ensure your account has this service-linked role, create an environment at least once using the console, and configure managed updates to be enabled before you create the environment.

Using the EB CLI

If you launch an environment using the the section called "**eb create**" command of the Elastic Beanstalk Command Line Interface (EB CLI) and don't specify a service role through the `--service-role` option, Elastic Beanstalk creates the default service role `aws-elasticbeanstalk-service-role`. If the default service role already exists, Elastic Beanstalk uses it for the new environment. The Elastic Beanstalk console also performs similar actions in these situations.

Unlike in the console, you can't specify a managed-updates service role when using an EB CLI command option. If you enable managed updates for your environment, you must set the managed-updates service role though configuration options. The following example enables managed updates and uses the default service role as a managed-updates service role.

Example .ebextensions/managed-platform-update.config

```
option_settings:
  aws:elasticbeanstalk:managedactions:
    ManagedActionsEnabled: true
    PreferredStartTime: "Tue:09:00"
    ServiceRoleForManagedUpdates: "aws-elasticbeanstalk-service-role"
  aws:elasticbeanstalk:managedactions:platformupdate:
    UpdateLevel: patch
    InstanceRefreshEnabled: true
```

Managing service roles using the Elastic Beanstalk API

When you use the `CreateEnvironment` action of the Elastic Beanstalk API to create an environment, specify a service role using the `ServiceRole` configuration option in the `aws:elasticbeanstalk:environment` namespace. For more information about using enhanced health monitoring with the Elastic Beanstalk API, see Using enhanced health reporting with the Elastic Beanstalk API.

In addition, if you enable managed platform updates for your environment, you can specify a managed-updates service role using the `ServiceRoleForManagedUpdates` option of the `aws:elasticbeanstalk:managedactions` namespace.

Using service-linked roles

A service-linked role is a unique type of service role that's predefined by Elastic Beanstalk to include all the permissions that the service requires to call other AWS services on your behalf. The service-linked role is associated with your account. Elastic Beanstalk creates it once, then reuses it when creating additional environments. For more information about using service-linked roles with Elastic Beanstalk environments, see Using service-linked roles for Elastic Beanstalk.

If you create an environment by using the Elastic Beanstalk API and don't specify a service role, Elastic Beanstalk creates a monitoring service-linked role for your account, if one doesn't already exist. Elastic Beanstalk uses this role for the new environment. You can also use IAM to create a monitoring service-linked role for your account in advance. After your account has this role, you can use it to create an environment using the Elastic Beanstalk API, the Elastic Beanstalk console, or the EB CLI.

If you enable managed platform updates for the environment and specify `AWSServiceRoleForElasticBeanstalkManagedUpdates` as the value for the `ServiceRoleForManagedUpdates` option of the `aws:elasticbeanstalk:managedactions` namespace, Elastic Beanstalk creates a managed-updates service-linked role for your account, if one doesn't already exist. Elastic Beanstalk uses the role to perform managed updates for the new environment.

Note

When Elastic Beanstalk tries to create the monitoring and managed-updates service-linked roles for your account when you create an environment, you must have the

`iam:CreateServiceLinkedRole` permission. If you don't have this permission, environment creation fails, and a message explaining the issue is displayed.
As an alternative, another user with permission to create service-linked roles can use IAM to create the service linked-role in advance. Using this method, you don't need the `iam:CreateServiceLinkedRole` permission to create your environment.

Verifying the default service role permissions

The permissions granted by your default service role can vary based on when they were created, the last time you launched an environment, and which client you used. In the IAM console, you can verify the permissions granted by the default service role.

To verify the default service role's permissions

1. In the IAM console, open the **Roles** page.
2. Choose **aws-elasticbeanstalk-service-role**.
3. On the **Permissions** tab, review the list of policies attached to the role.
4. To view the permissions that a policy grants, choose the policy.

Updating an out-of-date default service role

If the default service role lacks the required permissions, you can update it by creating a new environment in the Elastic Beanstalk environment management console.

Alternatively, you can manually add the managed policies to the default service role.

To add managed policies to the default service role

1. In the IAM console, open the **Roles** page .
2. Choose **aws-elasticbeanstalk-service-role**.
3. On the **Permissions** tab, choose **Attach policies**.
4. Enter **`AWSElasticBeanstalk`** to filter the policies.
5. Select the following policies, and then choose **Attach policy**:

   - `AWSElasticBeanstalkEnhancedHealth`
   - `AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy`

Adding permissions to the default service role

If your application includes configuration files that refer to AWS resources that permissions aren't included in the default service role for, Elastic Beanstalk might need additional permissions. These additional permissions are needed to resolve these references when it processes the configuration files during a managed update. If the permissions are missing, the update fails, and Elastic Beanstalk returns a message indicating which permissions it needs. Follow these steps to add permissions for additional services to the default service role in the IAM console.

To add additional policies to the default service role

1. In the IAM console, open the **Roles** page.
2. Choose **aws-elasticbeanstalk-service-role**.
3. On the **Permissions** tab, choose **Attach policies**.
4. Select the managed policy for the additional services that your application uses. For example, `AmazonAPIGatewayAdministrator` or `AmazonElasticFileSystemFullAccess`.
5. Choose **Attach policy**.

Creating a service role

If you can't use the default service role, create a service role.

To create a service role

1. In the IAM console, open the **Roles** page.
2. Choose **Create role**.
3. Under **AWS service**, choose **AWS Elastic Beanstalk**, and then select your use case.
4. Choose **Next: Permissions**.
5. Attach the `AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy` and `AWSElasticBeanstalkEnhancedHealth` managed policies and any additional policies that provide permissions that your application needs.
6. Choose **Next: Tags**.
7. (Optional) Add tags to the role.
8. Choose **Next: Review**.
9. Enter a name for the role.

10. Choose **Create role**.

Apply your custom service role when you create an environment either using the environment creation wizard or with the `--service-role` option for the `eb create` command.

Using service-linked roles for Elastic Beanstalk

AWS Elastic Beanstalk uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Elastic Beanstalk. Service-linked roles are predefined by Elastic Beanstalk and include all the permissions that the service requires to call other AWS services on your behalf.

Elastic Beanstalk defines a few types of service-linked roles:

- *Monitoring service-linked role* – Allows Elastic Beanstalk to monitor the health of running environments and publish health event notifications.
- *Maintenance service-linked role* – Allows Elastic Beanstalk to perform regular maintenance activities for your running environments.
- *Managed-updates service-linked role* – Allows Elastic Beanstalk to perform scheduled platform updates of your running environments.

Topics

The monitoring service-linked role

AWS Elastic Beanstalk uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Elastic Beanstalk. Service-linked roles are predefined by Elastic Beanstalk and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Elastic Beanstalk easier because you don't have to manually add the necessary permissions. Elastic Beanstalk defines the permissions of its service-linked roles, and unless defined otherwise, only Elastic Beanstalk can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Elastic Beanstalk resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Elastic Beanstalk

Elastic Beanstalk uses the service-linked role named **AWSServiceRoleForElasticBeanstalk** – Allows Elastic Beanstalk to monitor the health of running environments and publish health event notifications.

The AWSServiceRoleForElasticBeanstalk service-linked role trusts the following services to assume the role:

- `elasticbeanstalk.amazonaws.com`

The permissions policy of the AWSServiceRoleForElasticBeanstalk service-linked role contains all of the permissions that Elastic Beanstalk needs to complete actions on your behalf:

AllowCloudformationReadOperationsOnElasticBeanstalkStacks

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCloudformationReadOperationsOnElasticBeanstalkStacks",
            "Effect": "Allow",
            "Action": [
                "cloudformation:DescribeStackResource",
                "cloudformation:DescribeStackResources",
                "cloudformation:DescribeStacks"
            ],
            "Resource": [
                "arn:aws:cloudformation:*:*:stack/awseb-*",
                "arn:aws:cloudformation:*:*:stack/eb-*"
            ]
        },
        {
            "Sid": "AllowOperations",
            "Effect": "Allow",
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeAutoScalingInstances",
                "autoscaling:DescribeNotificationConfigurations",
                "autoscaling:DescribeScalingActivities",
                "autoscaling:PutNotificationConfiguration",
                "ec2:DescribeInstanceStatus",
                "ec2:AssociateAddress",
                "ec2:DescribeAddresses",
                "ec2:DescribeInstances",
                "ec2:DescribeSecurityGroups",
                "elasticloadbalancing:DescribeInstanceHealth",
                "elasticloadbalancing:DescribeLoadBalancers",
                "elasticloadbalancing:DescribeTargetHealth",
                "elasticloadbalancing:DescribeTargetGroups",
                "sqs:GetQueueAttributes",
                "sqs:GetQueueUrl",
                "sns:Publish"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the *IAM User Guide*.

Alternatively, you can use an AWS managed policy to provide full access to Elastic Beanstalk.

Creating a service-linked role for Elastic Beanstalk

You don't need to manually create a service-linked role. When you create an Elastic Beanstalk environment using the Elastic Beanstalk API and don't specify a service role, Elastic Beanstalk creates the service-linked role for you.

Important

If you were using the Elastic Beanstalk service before September 27, 2017, when it began supporting the AWSServiceRoleForElasticBeanstalk service-linked role, and your account needed it, then Elastic Beanstalk created the AWSServiceRoleForElasticBeanstalk role in your account. To learn more, see A New Role Appeared in My IAM Account.

When Elastic Beanstalk tries to create the AWSServiceRoleForElasticBeanstalk service-linked role for your account when you create an environment, you must have the `iam:CreateServiceLinkedRole` permission. If you don't have this permission, environment creation fails, and you see a message explaining the issue.

As an alternative, another user with permission to create service-linked roles can use IAM to pre-create the service linked-role in advance. You can then create your environment even without having the `iam:CreateServiceLinkedRole` permission.

You (or another user) can use the IAM console to create a service-linked role with the **Elastic Beanstalk** use case. In the IAM CLI or the IAM API, create a service-linked role with the `elasticbeanstalk.amazonaws.com` service name. For more information, see Creating a Service-Linked Role in the *IAM User Guide*. If you delete this service-linked role, you can use this same process to create the role again.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create an Elastic Beanstalk environment using the Elastic Beanstalk API and don't specify a service role, Elastic Beanstalk creates the service-linked role for you again.

Editing a service-linked role for Elastic Beanstalk

Elastic Beanstalk does not allow you to edit the AWSServiceRoleForElasticBeanstalk service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a Service-Linked Role in the *IAM User Guide*.

Deleting a service-linked role for Elastic Beanstalk

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first be sure that all Elastic Beanstalk environments are either using a different service role or are terminated.

Note

If the Elastic Beanstalk service is using the service-linked role when you try to terminate the environments, then the termination might fail. If that happens, wait for a few minutes and try the operation again.

To terminate an Elastic Beanstalk environment that uses the AWSServiceRoleForElasticBeanstalk (console)

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

See **eb terminate** for details about terminating an Elastic Beanstalk environment using the EB CLI.

See TerminateEnvironment for details about terminating an Elastic Beanstalk environment using the API.

Manually delete the service-linked role

Use the IAM console, the IAM CLI, or the IAM API to delete the AWSServiceRoleForElasticBeanstalk service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Supported regions for Elastic Beanstalk service-linked roles

Elastic Beanstalk supports using service-linked roles in all of the regions where the service is available. For more information, see AWS Elastic Beanstalk Endpoints and Quotas.

The maintenance service-linked role

AWS Elastic Beanstalk uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Elastic Beanstalk. Service-linked roles are predefined by Elastic Beanstalk and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Elastic Beanstalk easier because you don’t have to manually add the necessary permissions. Elastic Beanstalk defines the permissions of its service-linked roles, and unless defined otherwise, only Elastic Beanstalk can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Elastic Beanstalk resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Elastic Beanstalk

Elastic Beanstalk uses the service-linked role named **AWSServiceRoleForElasticBeanstalkMaintenance** – Allows Elastic Beanstalk to perform regular maintenance activities for your running environments.

The AWSServiceRoleForElasticBeanstalkMaintenance service-linked role trusts the following services to assume the role:

- `maintenance.elasticbeanstalk.amazonaws.com`

The permissions policy of the AWSServiceRoleForElasticBeanstalkMaintenance service-linked role contains all of the permissions that Elastic Beanstalk needs to complete actions on your behalf:

JSON

```
{
    "Version": "2012-10-17",
    "Statement":
        {
            "Sid":
 "AllowCloudformationChangeSetOperationsOnElasticBeanstalkStacks",
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateChangeSet",
                "cloudformation:DescribeChangeSet",
                "cloudformation:ExecuteChangeSet",
                "cloudformation:DeleteChangeSet",
                "cloudformation:ListChangeSets",
                "cloudformation:DescribeStacks"
            ],
            "Resource": [
                "arn:aws:cloudformation:*:*:stack/awseb-*",
                "arn:aws:cloudformation:*:*:stack/eb-*"
            ]
        }
}
```

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the *IAM User Guide*.

Alternatively, you can use an AWS managed policy to provide full access to Elastic Beanstalk.

Creating a service-linked role for Elastic Beanstalk

You don't need to manually create a service-linked role. When you create an Elastic Beanstalk environment using the Elastic Beanstalk API and don't specify an instance profile, Elastic Beanstalk creates the service-linked role for you.

Important

This service-linked role can appear in your account if you completed an action in another service that uses the features supported by this role. If you were using the Elastic Beanstalk service before April 18, 2019, when it began supporting the AWSServiceRoleForElasticBeanstalkMaintenance service-linked role, and your account needed it, then Elastic Beanstalk created the AWSServiceRoleForElasticBeanstalkMaintenance role in your account. To learn more, see A New Role Appeared in My IAM Account.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create an Elastic Beanstalk environment using the Elastic Beanstalk API and don't specify an instance profile, Elastic Beanstalk creates the service-linked role for you again.

Editing a service-linked role for Elastic Beanstalk

Elastic Beanstalk does not allow you to edit the AWSServiceRoleForElasticBeanstalkMaintenance service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a Service-Linked Role in the *IAM User Guide*.

Deleting a service-linked role for Elastic Beanstalk

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don’t have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first terminate any Elastic Beanstalk environments that uses the role.

Note

If the Elastic Beanstalk service is using the service-linked role when you try to terminate the environments, then the termination might fail. If that happens, wait for a few minutes and try the operation again.

To terminate an Elastic Beanstalk environment that uses the AWSServiceRoleForElasticBeanstalkMaintenance (console)

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Terminate environment**.
4. Use the on-screen dialog box to confirm environment termination.

See **eb terminate** for details about terminating an Elastic Beanstalk environment using the EB CLI.

See TerminateEnvironment for details about terminating an Elastic Beanstalk environment using the API.

Manually delete the service-linked role

Use the IAM console, the IAM CLI, or the IAM API to delete the AWSServiceRoleForElasticBeanstalkMaintenance service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Supported regions for Elastic Beanstalk service-linked roles

Elastic Beanstalk supports using service-linked roles in all of the regions where the service is available. For more information, see AWS Elastic Beanstalk Endpoints and Quotas.

The managed-updates service-linked role

AWS Elastic Beanstalk uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to Elastic Beanstalk. Service-linked roles are predefined by Elastic Beanstalk and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Elastic Beanstalk easier because you don’t have to manually add the necessary permissions. Elastic Beanstalk defines the permissions of its service-linked roles, and unless defined otherwise, only Elastic Beanstalk can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Elastic Beanstalk resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Elastic Beanstalk

Elastic Beanstalk uses the service-linked role named **AWSServiceRoleForElasticBeanstalkManagedUpdates** – Allows Elastic Beanstalk to perform scheduled platform updates of your running environments.

The AWSServiceRoleForElasticBeanstalkManagedUpdates service-linked role trusts the following services to assume the role:

- `managedupdates.elasticbeanstalk.amazonaws.com`

The managed policy **AWSElasticBeanstalkManagedUpdatesServiceRolePolicy** allows the AWSServiceRoleForElasticBeanstalkManagedUpdates service-linked role all of the permissions that Elastic Beanstalk needs to complete managed update actions on your behalf. To view the managed policy content, see the AWSElasticBeanstalkManagedUpdatesServiceRolePolicy page in the *AWS Managed Policy Reference Guide*.

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the *IAM User Guide*.

Alternatively, you can use an AWS managed policy to provide full access to Elastic Beanstalk.

Creating a service-linked role for Elastic Beanstalk

You don't need to manually create a service-linked role. When you create an Elastic Beanstalk environment using the Elastic Beanstalk API, enable managed updates, and specify `AWSServiceRoleForElasticBeanstalkManagedUpdates` as the value for the `ServiceRoleForManagedUpdates` option of the `aws:elasticbeanstalk:managedactions` namespace, Elastic Beanstalk creates the service-linked role for you.

When Elastic Beanstalk tries to create the AWSServiceRoleForElasticBeanstalkManagedUpdates service-linked role for your account when you create an environment, you must have the `iam:CreateServiceLinkedRole` permission. If you don't have this permission, environment creation fails, and you see a message explaining the issue.

As an alternative, another user with permission to create service-linked roles can use IAM to pre-create the service linked-role in advance. You can then create your environment even without having the `iam:CreateServiceLinkedRole` permission.

You (or another user) can use the IAM console to create a service-linked role with the **Elastic Beanstalk Managed Updates** use case. In the IAM CLI or the IAM API, create a service-linked role with the `managedupdates.elasticbeanstalk.amazonaws.com` service name. For more information, see Creating a Service-Linked Role in the *IAM User Guide*. If you delete this service-linked role, you can use this same process to create the role again.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create an Elastic Beanstalk environment using the Elastic Beanstalk API, enable managed updates, and specify `AWSServiceRoleForElasticBeanstalkManagedUpdates` as the value for the `ServiceRoleForManagedUpdates` option of the `aws:elasticbeanstalk:managedactions` namespace, Elastic Beanstalk creates the service-linked role for you again.

Editing a service-linked role for Elastic Beanstalk

Elastic Beanstalk does not allow you to edit the AWSServiceRoleForElasticBeanstalkManagedUpdates service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a Service-Linked Role in the *IAM User Guide*.

Deleting a service-linked role for Elastic Beanstalk

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don’t have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first be sure that Elastic Beanstalk environments with managed updates enabled are either using a different service role or are terminated.

Note

If the Elastic Beanstalk service is using the service-linked role when you try to terminate the environments, then the termination might fail. If that happens, wait for a few minutes and try the operation again.

To terminate an Elastic Beanstalk environment that uses the AWSServiceRoleForElasticBeanstalkManagedUpdates (console)

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Terminate Environment**.
4. Use the on-screen dialog box to confirm environment termination.

See **eb terminate** for details about terminating an Elastic Beanstalk environment using the EB CLI.

See TerminateEnvironment for details about terminating an Elastic Beanstalk environment using the API.

Manually delete the service-linked role

Use the IAM console, the IAM CLI, or the IAM API to delete the AWSServiceRoleForElasticBeanstalkManagedUpdates service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Supported Regions for Elastic Beanstalk service-linked roles

Elastic Beanstalk supports using service-linked roles in all of the regions where the service is available. For more information, see AWS Elastic Beanstalk Endpoints and Quotas.

Managing Elastic Beanstalk user policies

AWS Elastic Beanstalk provides two managed policies that enable you to assign full access or read-only access to all resources that Elastic Beanstalk manages. You can attach the policies to AWS Identity and Access Management (IAM) users or groups, or to roles assumed by your users.

Managed user policies

- **AdministratorAccess-AWSElasticBeanstalk** – Gives the user full administrative permissions to create, modify, and delete Elastic Beanstalk applications, application versions, configuration settings, environments, and their underlying resources. To view the managed policy content, see the AdministratorAccess-AWSElasticBeanstalk page in the *AWS Managed Policy Reference Guide*.
- **AWSElasticBeanstalkReadOnly** – Allows the user to view applications and environments, but not to perform operations that modify them. It provides read-only access to all Elastic Beanstalk resources, and to other AWS resources that the Elastic Beanstalk console retrieves. Note that read-only access does not enable actions such as downloading Elastic Beanstalk logs so that you can read them. This is because the logs are staged in the Amazon S3 bucket, where Elastic Beanstalk would require write permission. See the example at the end of this topic for information on how to enable access to Elastic Beanstalk logs. To view the managed policy content, see the AWSElasticBeanstalkReadOnly page in the *AWS Managed Policy Reference Guide*.

Important

Elastic Beanstalk managed policies don't provide granular permissions—they grant all permissions that are potentially needed for working with Elastic Beanstalk applications. In some cases you may wish to restrict the permissions of our managed policies further. For an example of one use case, see Preventing cross-environment Amazon S3 bucket access. Our managed policies also don't cover permissions to custom resources that you might add to your solution, and that aren't managed by Elastic Beanstalk. To implement more granular permissions, minimum required permissions, or custom resource permissions, use custom policies.

Deprecated managed policies

Previously, Elastic Beanstalk supported two other managed user policies, **AWSElasticBeanstalkFullAccess** and **AWSElasticBeanstalkReadOnlyAccess**. We plan on retiring these previous policies. You might still be able to see and use them in the IAM console. Nevertheless, we recommend that you transition to using the new managed user policies, and add custom policies to grant permissions to custom resources, if you have any.

Policies for integration with other services

We also provide more granular policies that allow you to integrate your environment with other services, if you prefer to use those.

- **AWSElasticBeanstalkRoleCWL** – Allows an environment to manage Amazon CloudWatch Logs log groups.
- **AWSElasticBeanstalkRoleRDS** – Allows an environment to integrate an Amazon RDS instance.
- **AWSElasticBeanstalkRoleWorkerTier** – Allows a worker environment tier to create an Amazon DynamoDB table and an Amazon SQS queue.
- **AWSElasticBeanstalkRoleECS** – Allows a multicontainer Docker environment to manage Amazon ECS clusters.
- **AWSElasticBeanstalkRoleCore** – Allows core operations of a web service environment.
- **AWSElasticBeanstalkRoleSNS** – Allows an environment to enable Amazon SNS topic integration.

To see the JSON source for a specific managed policy, see the *AWS Managed Policy Reference Guide*.

Controlling access with managed policies

You can use managed policies to grant full access or read-only access to Elastic Beanstalk. Elastic Beanstalk updates these policies automatically when additional permissions are required to access new features.

To apply a managed policy to IAM users or groups

1. Open the **Policies** page in the IAM console.
2. In the search box, type **`AWSElasticBeanstalk`** to filter the policies.
3. In the list of policies, select the check box next to **AWSElasticBeanstalkReadOnly** or **AdministratorAccess-AWSElasticBeanstalk**.
4. Choose **Policy actions**, and then choose **Attach**.
5. Select one or more users and groups to attach the policy to. You can use the **Filter** menu and the search box to filter the list of principal entities.
6. Choose **Attach policy**.

Creating a custom user policy

You can create your own IAM policy to allow or deny specific Elastic Beanstalk API actions on specific Elastic Beanstalk resources, and to control access to custom resources that aren't managed by Elastic Beanstalk. For more information about attaching a policy to a user or group, see Working with Policies in the *IAM User Guide*. For details about creating a custom policy, see Creating IAM Policies in the *IAM User Guide*.

Note

While you can restrict how a user interacts with Elastic Beanstalk APIs, there is not currently an effective way to prevent users who have permission to create the necessary underlying resources from creating other resources in Amazon EC2 and other services.
Think of these policies as an effective way to distribute Elastic Beanstalk responsibilities, not as a way to secure all underlying resources.

Important

If you have custom policies assigned to an Elastic Beanstalk service role, it's important that you assign it the proper permissions for launch templates. Otherwise you may not have the required permissions to update an environment or launch a new one. For more information, see Required permissions for launch templates.

An IAM policy contains policy statements that describe the permissions that you want to grant. When you create a policy statement for Elastic Beanstalk, you need to understand how to use the following four parts of a policy statement:

- **Effect** specifies whether to allow or deny the actions in the statement.
- **Action** specifies the API operations that you want to control. For example, use `elasticbeanstalk:CreateEnvironment` to specify the `CreateEnvironment` operation. Certain operations, such as creating an environment, require additional permissions to perform those actions. For more information, see Resources and conditions for Elastic Beanstalk actions.

Note

To use the [UpdateTagsForResource](#) API operation, specify one of the following two virtual actions (or both) instead of the API operation name:

`elasticbeanstalk:AddTags`

Controls permission to call `UpdateTagsForResource` and pass a list of tags to add in the `TagsToAdd` parameter.

`elasticbeanstalk:RemoveTags`

Controls permission to call `UpdateTagsForResource` and pass a list of tag keys to remove in the `TagsToRemove` parameter.

- **Resource** specifies the resources that you want to control access to. To specify Elastic Beanstalk resources, list the [Amazon Resource Name](#) (ARN) of each resource.
- (optional) **Condition** specifies restrictions on the permission granted in the statement. For more information, see [Resources and conditions for Elastic Beanstalk actions](#).

The following sections demonstrate a few cases in which you might consider a custom user policy.

Enabling limited Elastic Beanstalk environment creation

The policy in the following example enables a user to call the `CreateEnvironment` action to create an environment whose name begins with **`Test`** with the specified application and application version.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid":"CreateEnvironmentPerm",
      "Action": [
        "elasticbeanstalk:CreateEnvironment"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My First
 Elastic Beanstalk Application/Test*"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
east-2:123456789012:application/My First Elastic Beanstalk Application"],
          "elasticbeanstalk:FromApplicationVersion":
 ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My First
 Elastic Beanstalk Application/First Release"]
        }
      }
    },
    {
      "Sid":"AllNonResourceCalls",
      "Action":[
        "elasticbeanstalk:CheckDNSAvailability",
        "elasticbeanstalk:CreateStorageLocation"
      ],
      "Effect":"Allow",
      "Resource":[
        "*"
      ]
    }
  ]
}
```

The above policy shows how to grant limited access to Elastic Beanstalk operations. In order to actually launch an environment, the user must have permission to create the AWS resources that power the environment as well. For example, the following policy grants access to the default set of resources for a web server environment:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:*",
        "ecs:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "cloudformation:*",
        "sqs:*"
        ],
      "Resource": "*"
    }
  ]
}
```

Enabling access to Elastic Beanstalk logs stored in Amazon S3

The policy in the following example enables a user to pull Elastic Beanstalk logs, stage them in Amazon S3, and retrieve them.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:DeleteObject",
        "s3:GetObjectAcl",
        "s3:PutObjectAcl"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::elasticbeanstalk-*"
    }
  ]
}
```

Note

To restrict these permissions to only the logs path, use the following resource format.

```
"arn:aws:s3:::elasticbeanstalk-us-east-2-123456789012/resources/environments/
logs/*"
```

Enabling management of a specific Elastic Beanstalk application

The policy in the following example enables a user to manage environments and other resources within one specific Elastic Beanstalk application. The policy denies Elastic Beanstalk actions on resources of other applications, and also denies creation and deletion of Elastic Beanstalk applications.

Note

The policy doesn't deny access to any resources through other services. It demonstrates an effective way to distribute responsibilities for managing Elastic Beanstalk applications among different users, not as a way to secure the underlying resources.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "elasticbeanstalk:CreateApplication",
        "elasticbeanstalk:DeleteApplication"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "elasticbeanstalk:CreateApplicationVersion",
        "elasticbeanstalk:CreateConfigurationTemplate",
        "elasticbeanstalk:CreateEnvironment",
        "elasticbeanstalk:DeleteApplicationVersion",
        "elasticbeanstalk:DeleteConfigurationTemplate",
        "elasticbeanstalk:DeleteEnvironmentConfiguration",
        "elasticbeanstalk:DescribeApplicationVersions",
        "elasticbeanstalk:DescribeConfigurationOptions",
        "elasticbeanstalk:DescribeConfigurationSettings",
        "elasticbeanstalk:DescribeEnvironmentResources",
        "elasticbeanstalk:DescribeEnvironments",
        "elasticbeanstalk:DescribeEvents",
        "elasticbeanstalk:DeleteEnvironmentConfiguration",
        "elasticbeanstalk:RebuildEnvironment",
        "elasticbeanstalk:RequestEnvironmentInfo",
        "elasticbeanstalk:RestartAppServer",
        "elasticbeanstalk:RetrieveEnvironmentInfo",
        "elasticbeanstalk:SwapEnvironmentCNAMEs",
        "elasticbeanstalk:TerminateEnvironment",
        "elasticbeanstalk:UpdateApplicationVersion",
        "elasticbeanstalk:UpdateConfigurationTemplate",
        "elasticbeanstalk:UpdateEnvironment",
        "elasticbeanstalk:RetrieveEnvironmentInfo",
        "elasticbeanstalk:ValidateConfigurationSettings"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringNotEquals": {
          "elasticbeanstalk:InApplication": [
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/
myapplication"
          ]
        }
      }
    }
  ]
}
```

Amazon resource name format for Elastic Beanstalk

You specify a resource for an IAM policy using that resource's Amazon Resource Name (ARN). For Elastic Beanstalk, the ARN has the following format.

```
arn:aws:elasticbeanstalk:region:account-id:resource-type/resource-path
```

Where:

- *`region`* is the region the resource resides in (for example, **`us-west-2`**).
- *`account-id`* is the AWS account ID, with no hyphens (for example, **`123456789012`**)
- *`resource-type`* identifies the type of the Elastic Beanstalk resource—for example, `environment`. See the table below for a list of all Elastic Beanstalk resource types.
- *`resource-path`* is the portion that identifies the specific resource. An Elastic Beanstalk resource has a path that uniquely identifies that resource. See the table below for the format of the resource path for each resource type. For example, an environment is always associated with an application. The resource path for the environment **`myEnvironment`** in the application **`myApp`** would look like this:

  ```
  myApp/myEnvironment
  ```

Elastic Beanstalk has several types of resources you can specify in a policy. The following table shows the ARN format for each resource type and an example.

Resource type	**Format for ARN**
`application`	`arn:aws:elasticbeanstalk:` *`region`*`:`*`account-id`* `:application/` *`application-name`* Example: **`arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App`**
`applicationversion`	`arn:aws:elasticbeanstalk:` *`region`*`:`*`account-id`* `:applicationversion/` *`application-name`* `/`*`version-label`* Example: **`arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version`**
`configurationtemplate`	`arn:aws:elasticbeanstalk:` *`region`*`:`*`account-id`* `:configurationtemplate/` *`application-name`* `/`*`template-name`*

Resource type	Format for ARN
	Example: **`arn:aws:elasticbeanstalk:us-east-2:123456789012:configurationtemplate/My App/My Template`**
`environment`	`arn:aws:elasticbeanstalk:` *`region`*`:`*`account-id`* `:environment/` *`application-name`* `/`*`environment-name`* Example: **`arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/MyEnvironment`**
`platform`	`arn:aws:elasticbeanstalk:` *`region`*`:`*`account-id`* `:platform/` *`platform-name`* `/`*`platform-version`* Example: **`arn:aws:elasticbeanstalk:us-east-2:123456789012:platform/MyPlatform/1.0`**
`solutionstack`	`arn:aws:elasticbeanstalk:` *`region`*`::solutionstack/` *`solutionstack-name`* Example: **`arn:aws:elasticbeanstalk:us-east-2::solutionstack/32bit Amazon Linux running Tomcat 7`**

An environment, application version, and configuration template are always contained within a specific application. You'll notice that these resources all have an application name in their resource path so that they are uniquely identified by their resource name and the containing application. Although solution stacks are used by configuration templates and environments, solution stacks are not specific to an application or AWS account and do not have the application or AWS account in their ARNs.

Resources and conditions for Elastic Beanstalk actions

This section describes the resources and conditions that you can use in policy statements to grant permissions that allow specific Elastic Beanstalk actions to be performed on specific Elastic Beanstalk resources.

Conditions enable you to specify permissions to resources that the action needs to complete. For example, when you can call the `CreateEnvironment` action, you must also specify the application version to deploy as well as the application that contains that application name. When you set permissions for the `CreateEnvironment` action, you specify the application and application version that you want the action to act upon by using the `InApplication` and `FromApplicationVersion` conditions.

In addition, you can specify the environment configuration with a solution stack (`FromSolutionStack`) or a configuration template (`FromConfigurationTemplate`). The following policy statement allows the `CreateEnvironment` action to create an environment with the name **`myenv`** (specified by `Resource`) in the application **`My App`** (specified by the `InApplication` condition) using the application version **`My Version`** (`FromApplicationVersion`) with a **`32bit Amazon Linux running Tomcat 7`** configuration (`FromSolutionStack`):

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:CreateEnvironment"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/
myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
east-2:123456789012:application/My App"],
          "elasticbeanstalk:FromApplicationVersion":
 ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My
 Version"],
          "elasticbeanstalk:FromSolutionStack": ["arn:aws:elasticbeanstalk:us-
east-2::solutionstack/32bit Amazon Linux running Tomcat 7"]
        }
      }
    }
  ]
}
```

Note

Most condition keys mentioned in this topic are specific to Elastic Beanstalk, and their names contain the `elasticbeanstalk:` prefix. For brevity, we omit this prefix from the condition key names when we mention them in the following sections. For example, we mention `InApplication` instead of its full name `elasticbeanstalk:InApplication`. In contrast, we mention a few condition keys used across AWS services, and we include their `aws:` prefix to highlight the exception.
Policy examples always show full condition key names, including the prefix.

Sections

- Policy information for Elastic Beanstalk actions
- Condition keys for Elastic Beanstalk actions

Policy information for Elastic Beanstalk actions

The following table lists all Elastic Beanstalk actions, the resource that each action acts upon, and the additional contextual information that can be provided using conditions.

Policy information for Elastic Beanstalk actions, including resources, conditions, examples, and dependencies

Resource	Conditions	Example statement
Action: `AbortEnvironmentUpdate`		

`application`
 	
`environment`
	`aws:ResourceTag/ key-name` (Optional)
	
	`aws:TagKeys` (Optional)
		The following policy allows a user to abort environment update operations on environments in an application named `My App`.
		
		JSON
		
		`{`
		`  "Version": "2012-10-17",`

Resource	Conditions	Example statement
		```
		  "Statement": [
		    {
		      "Action": [
		        "elasticbeanstalk:AbortEnvi
		ronmentUpdate"
		      ],
		      "Effect": "Allow",
		      "Resource": [
		        "arn:aws:elasticbeanstalk:us-
		east-2:123456789012:application/My
		 App"
		      ]
		    }
		  ]
		}
		```
Action: CheckDNSAvailability		

`"*"`	N/A	JSON
		```
		{
		  "Version": "2012-10-17",
		  "Statement": [
		    {
		      "Action": [
		        "elasticbeanstalk:CheckDNSA
		vailability"
		      ],
		      "Effect": "Allow",
		      "Resource": "*"
		    }
		  ]
		}
		```

Action: ComposeEnvironments		

Resource	Conditions	Example statement
`application`	`aws:ResourceTag/ key-name` (Optional) `aws:TagKeys` (Optional)	The following policy allows a user to compose environments that belong to an application named `My App`.
Action: `CreateApplication`		
`application`	`aws:RequestTag/ key-name` (Optional) `aws:TagKeys` (Optional)	This example allows the `CreateApplication` action to create applications whose names begin with **`DivA`**: JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:CreateApplication"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/DivA*"
      ]
    }
  ]
}
```

Action: `CreateApplicationVersion`

Resource	Conditions	Example statement
`applicationversion`	`InApplication` `aws:RequestTag/` *`key-name`* (Optional) `aws:TagKeys` (Optional)	This example allows the `CreateApplicationVersion` action to create application versions with any name (*****) in the application **`My App`**: JSON `{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:CreateApplicationVersion"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/*"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }`

Action: `CreateConfigurationTemplate`

Resource	Conditions	Example statement

`configurationtemplate`	`InApplication`
	
	`FromApplication`
	
	`FromApplicationVersion`
	
	`FromConfigurationTemplate`
	
	`FromEnvironment`
	
	`FromSolutionStack`
	
	`aws:RequestTag/` *`key-name`* (Optional)
	
	`aws:TagKeys` (Optional)
		The following policy allows the `CreateConfigurationTemplate` action to create configuration templates whose name begins with **My Template** (`My Template*`) in the application **My App**:
		
		JSON
		
		`{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:CreateConfigurationTemplate"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:configurationtemplate/My App/My Template*"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"], "elasticbeanstalk:FromSolutionStack": ["arn:aws:elasticbeanstalk:us-east-2::solutionstack/32bit Amazon Linux running Tomcat 7"] } } }] }`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:CreateConfigurationTemplate"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:configurationtemplate/My App/My Template*"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"],
          "elasticbeanstalk:FromSolutionStack": ["arn:aws:elasticbeanstalk:us-east-2::solutionstack/32bit Amazon Linux running Tomcat 7"]
        }
      }
    }
  ]
}
```

Resource	Conditions	Example statement
Action: `CreateEnvironment`		

Resource	Conditions	Example statement

environment	InApplication
	
	FromApplicationVersion
	
	FromConfigurationTemplate
	
	FromSolutionStack
	
	aws:RequestTag/ *key-name* (Optional)
	
	aws:TagKeys (Optional)
		The following policy allows the CreateEnvironment action to create an environment whose name is **myenv** in the application **My App** and using the solution stack **32bit Amazon Linux running Tomcat 7**:
		
		JSON
		
		`{`
		`  "Version": "2012-10-17",`
		`  "Statement": [`
		`    {`
		`      "Action": [`
		`        "elasticbeanstalk:CreateEnvironment"`
		`      ],`
		`      "Effect": "Allow",`
		`      "Resource": [`
		`        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"`
		`      ],`
		`      "Condition": {`
		`        "StringEquals": {`
		`          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"],`
		`          "elasticbeanstalk:FromApplicationVersion": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version"],`
		`          "elasticbeanstalk:FromSolutionStack": ["arn:aws:elasticbeanstalk:us-east-2::solutionstack/32bit Amazon Linux running Tomcat 7"]`
		`        }`
		`      }`
		`    }`

Resource	Conditions	Example statement
		```
		  ]
		}
		```
Action: `CreatePlatformVersion`		

`platform`	`aws:RequestTag/ key-name` (Optional)
	
	`aws:TagKeys` (Optional)
		This example allows the `CreatePlatformVersion` action to create platform versions targeting the `us-east-2` region, whose names begin with **`us-east-2_`**:
		
		JSON
		```
		{
		  "Version": "2012-10-17",
		  "Statement": [
		    {
		      "Action": [
		        "elasticbeanstalk:CreatePlatformVersion"
		      ],
		      "Effect": "Allow",
		      "Resource": [
		        "arn:aws:elasticbeanstalk:us-east-2:123456789012:platform/us-east-2_*"
		      ]
		    }
		  ]
		}
		```

Action: `CreateStorageLocation`		

Resource	Conditions	Example statement

`"*"`	N/A	JSON
		```
		{
		  "Version": "2012-10-17",
		  "Statement": [
		    {
		      "Action": [
		        "elasticbeanstalk:CreateSto
		rageLocation"
		      ],
		      "Effect": "Allow",
		      "Resource": "*"
		    }
		  ]
		}
		```

Action: `DeleteApplication`

Resource	Conditions	Example statement
`application`	`aws:ResourceTag/` *`key-name`* (Optional) `aws:TagKeys` (Optional)	The following policy allows the `DeleteApplication` action to delete the application **`My App`**: JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": [ "elasticbeanstalk:DeleteApplication" ], "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App" ] } ] }`

Action: `DeleteApplicationVersion`

Resource	Conditions	Example statement
applicati onversion	InApplica tion aws:Resou rceTag/ *key-name* (Optional) aws:TagKeys (Optional)	The following policy allows the DeleteApp licationVersion action to delete an application version whose name is **My Version** in the applicati on **My App**: JSON `{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:DeleteApplicationVersion"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:DeleteApp
licationVersion"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:u
s-east-2:123456789012:applicationver
sion/My App/My Version"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplica
tion": ["arn:aws:elasticbeanstalk:
us-east-2:123456789012:application/M
y App"]
        }
      }
    }
  ]
}
```

Action: DeleteConfigurationTemplate

Resource	Conditions	Example statement
`configurationtemplate`	`InApplication` (Optional) `aws:ResourceTag/ key-name` (Optional) `aws:TagKeys` (Optional)	The following policy allows the `DeleteConfigurationTemplate` action to delete a configuration template whose name is **`My Template`** in the application **`My App`**. Specifying the application name as a condition is optional. JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": [ "elasticbeanstalk:DeleteConfigurationTemplate" ], "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:configurationtemplate/My App/My Template" ] } ] }`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:DeleteCon
figurationTemplate"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:u
s-east-2:123456789012:configurationt
emplate/My App/My Template"
      ]
    }
  ]
}
```

Action: `DeleteEnvironmentConfiguration`

Resource	Conditions	Example statement
`environment`	`InApplication` (Optional)	The following policy allows the `DeleteEnvironmentConfiguration` action to delete a draft configuration for the environment **myenv** in the application **My App**. Specifying the application name as a condition is optional. JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": [ "elasticbeanstalk:DeleteEnvironmentConfiguration" ], "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv" ] } ] }`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:DeleteEnv
ironmentConfiguration"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-
east-2:123456789012:environment/My
 App/myenv"
      ]
    }
  ]
}
```

Action: `DeletePlatformVersion`

Resource	Conditions	Example statement
`platform`	`aws:ResourceTag/ key-name` (Optional) `aws:TagKeys` (Optional)	The following policy allows the `DeletePlatformVersion` action to delete platform versions targeting the `us-east-2` region, whose names begin with **`us-east-2_`** : JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": [ "elasticbeanstalk:DeletePlatformVersion" ], "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:platform/us-east-2_*" ] } ] }`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:DeletePla
tformVersion"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-
east-2:123456789012:platform/us-ea
st-2_*"
      ]
    }
  ]
}
```

Action: `DescribeApplications`

Resource	Conditions	Example statement
`application`	`aws:ResourceTag/ key-name` (Optional) `aws:TagKeys` (Optional)	The following policy allows the `DescribeApplications` action to describe the application My App. JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": [ "elasticbeanstalk:DescribeApplications" ], "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App" ] } ] }`

Action: `DescribeApplicationVersions`

Resource	Conditions	Example statement
applicationversion	InApplication (Optional) aws:ResourceTag/ *key-name* (Optional) aws:TagKeys (Optional)	The following policy allows the DescribeApplicationVersions action to describe the application version **My Version** in the application **My App**. Specifying the application name as a condition is optional. JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": [ "elasticbeanstalk:DescribeApplicationVersions" ], "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version" ] } ] }`

Action: DescribeConfigurationOptions

Resource	Conditions	Example statement

`environment`
 	
`configurationtemplate`
 	
`solutionstack`
	`InApplication` (Optional)
	
	`aws:ResourceTag/ key-name` (Optional)
	
	`aws:TagKeys` (Optional)
		The following policy allows the `DescribeConfigurationOptions` action to describe the configuration options for the environment **myenv** in the application **My App**. Specifying the application name as a condition is optional.
		
		JSON
		
		`{ "Version": "2012-10-17", "Statement": [ { "Action": "elasticbeanstalk:DescribeConfigurationOptions", "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv" ] } ] }`

Action: `DescribeConfigurationSettings`

Resource	Conditions	Example statement

`environment`
 	
`configurationtemplate`
	`InApplication` (Optional)
	
	`aws:ResourceTag/ key-name` (Optional)
	
	`aws:TagKeys` (Optional)
		The following policy allows the `DescribeConfigurationSettings` action to describe the configuration settings for the environment **myenv** in the application **My App**. Specifying the application name as a condition is optional.
		
		JSON
		
		`{ "Version": "2012-10-17", "Statement": [ { "Action": "elasticbeanstalk: DescribeConfigurationSettings", "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv" ] } ] }`

Action: `DescribeEnvironmentHealth`

Resource	Conditions	Example statement
`environment`	`aws:ResourceTag/` *`key-name`* (Optional) `aws:TagKeys` (Optional)	The following policy allows use of `DescribeEnvironmentHealth` to retrieve health information for an environment named **`myenv`**. JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": "elasticbeanstalk:DescribeEnvironmentHealth", "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv" ] } ] }`

Action: [`DescribeEnvironmentResources`]()

Resource	Conditions	Example statement

`environment`	`InApplication` (Optional)
	
	`aws:ResourceTag/ key-name` (Optional)
	
	`aws:TagKeys` (Optional)
		The following policy allows the `DescribeEnvironmentResources` action to return list of AWS resources for the environment **myenv** in the application **My App**. Specifying the application name as a condition is optional.
		
		JSON
		
		`{`
		`  "Version": "2012-10-17",`
		`  "Statement": [`
		`    {`
		`      "Action": "elasticbeanstalk:DescribeEnvironmentResources",`
		`      "Effect": "Allow",`
		`      "Resource": [`
		`        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"`
		`      ]`
		`    }`
		`  ]`
		`}`

Action: `DescribeEnvironments`

Resource	Conditions	Example statement
`environment`	`InApplication` (Optional) `aws:ResourceTag/` *`key-name`* (Optional) `aws:TagKeys` (Optional)	The following policy allows the `DescribeEnvironments` action to describe the environments **`myenv`** and **`myotherenv`** in the application **`My App`**. Specifying the application name as a condition is optional. JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": "elasticbeanstalk:DescribeEnvironments", "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv", "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App2/myotherenv" ] } ] }`

Action: DescribeEvents

Resource	Conditions	Example statement
application applicati onversion configura tiontempl ate environment	InApplica tion aws:Resou rceTag/ *key-name* (Optional) aws:TagKeys (Optional)	The following policy allows the DescribeEvents action to list event descriptions for the environment **myenv** and the application version **My Version** in the application **My App**. JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "elasticbeanstalk:
DescribeEvents",
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-
east-2:123456789012:environment/My
 App/myenv",
        "arn:aws:elasticbeanstalk:u
s-east-2:123456789012:applicationver
sion/My App/My Version"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplica
tion": ["arn:aws:elasticbeanstalk:
us-east-2:123456789012:application/M
y App"]
        }
      }
    }
  ]
}
```

Action: DescribeInstancesHealth

Resource	Conditions	Example statement
`environment`	N/A	The following policy allows use of `DescribeInstancesHealth` to retrieve health information for instances in an environment named **myenv**. JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": "elasticbeanstalk:DescribeInstancesHealth", "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv" ] } ] }`

Action: `DescribePlatformVersion`

Resource	Conditions	Example statement
`platform`	`aws:ResourceTag/ key-name` (Optional) `aws:TagKeys` (Optional)	The following policy allows the `DescribePlatformVersion` action to describe platform versions targeting the `us-east-2` region, whose names begin with **`us-east-2_`** : JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": [ "elasticbeanstalk:DescribePlatformVersion" ], "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:platform/us-east-2_*" ] } ] }`

Action: `ListAvailableSolutionStacks`

Resource	Conditions	Example statement
solutionstack	N/A	The following policy allows the `ListAvailableSolutionStacks` action to return only the solution stack **`32bit Amazon Linux running Tomcat 7`**. JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": [ "elasticbeanstalk:ListAvailableSolutionStacks" ], "Effect": "Allow", "Resource": "arn:aws:elasticbeanstalk:us-east-2::solutionstack/32bit Amazon Linux running Tomcat 7" } ] }`

Action: `ListPlatformVersions`

Resource	Conditions	Example statement
`platform`	`aws:RequestTag/` *`key-name`* (Optional) `aws:TagKeys` (Optional)	This example allows the `CreatePlatformVersion` action to create platform versions targeting the `us-east-2` region, whose names begin with **`us-east-2_`**: JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": [ "elasticbeanstalk:ListPlatformVersions" ], "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:platform/us-east-2_*" ] } ] }`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:ListPlatf
ormVersions"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-
east-2:123456789012:platform/us-ea
st-2_*"
      ]
    }
  ]
}
```

Action: `ListTagsForResource`

Resource	Conditions	Example statement

`application`
 	
`applicationversion`
 	
`configurationtemplate`
 	
`environment`
 	
`platform`
	`aws:ResourceTag/` *`key-name`* (Optional)
	
	`aws:TagKeys` (Optional)
		The following policy allows the `ListTagsForResource` action to list tags of existing resources only if they have a tag named `stage` with the value `test`:
		
		JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:ListTagsForResource"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/stage": ["test"]
        }
      }
    }
  ]
}
```

Action: RebuildEnvironment

Resource	Conditions	Example statement

`environment`	`InApplication`
	
	`aws:ResourceTag/ key-name` (Optional)
	
	`aws:TagKeys` (Optional)
		The following policy allows the `RebuildEnvironment` action to rebuild the environment **`myenv`** in the application **`My App`**.
		
		JSON
		
		`{`
		`  "Version": "2012-10-17",`
		`  "Statement": [`
		`    {`
		`      "Action": [`
		`        "elasticbeanstalk:RebuildEnvironment"`
		`      ],`
		`      "Effect": "Allow",`
		`      "Resource": [`
		`        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"`
		`      ],`
		`      "Condition": {`
		`        "StringEquals": {`
		`          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"]`
		`        }`
		`      }`
		`    }`
		`  ]`
		`}`

Action: `RequestEnvironmentInfo`

Resource	Conditions	Example statement
`environment`	`InApplication` `aws:ResourceTag/ key-name` (Optional) `aws:TagKeys` (Optional)	The following policy allows the `RequestEnvironmentInfo` action to compile information about the environment **myenv** in the application **My App**. JSON `{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:RequestEnvironmentInfo"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }`

Action: `RestartAppServer`

Resource	Conditions	Example statement
`environment`	`InApplication`	The following policy allows the `RestartAppServer` action to restart the application container server for the environment **`myenv`** in the application **`My App`**. JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:RestartAppServer"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"]
        }
      }
    }
  ]
}
```

Action: `RetrieveEnvironmentInfo`

Resource	Conditions	Example statement

environment	InApplication
	
	aws:ResourceTag/ *key-name* (Optional)
	
	aws:TagKeys (Optional)
		The following policy allows the RetrieveEnvironmentInfo action to retrieve the compiled information for the environment **myenv** in the application **My App**.
		
		JSON
		
		`{`
		`  "Version": "2012-10-17",`
		`  "Statement": [`
		`    {`
		`      "Action": [`
		`        "elasticbeanstalk:RetrieveEnvironmentInfo"`
		`      ],`
		`      "Effect": "Allow",`
		`      "Resource": [`
		`        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"`
		`      ],`
		`      "Condition": {`
		`        "StringEquals": {`
		`          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"]`
		`        }`
		`      }`
		`    }`
		`  ]`
		`}`

Action: SwapEnvironmentCNAMEs

Resource	Conditions	Example statement

`environment`	`InApplication` (Optional)
	
	`FromEnvironment` (Optional)
		The following policy allows the `SwapEnvironmentCNAMEs` action to swap the CNAMEs for the environments **`mysrcenv`** and **`mydestenv`** .
		
		JSON
		```
		{
		  "Version": "2012-10-17",
		  "Statement": [
		    {
		      "Action": [
		        "elasticbeanstalk:SwapEnvironmentCNAMEs"
		      ],
		      "Effect": "Allow",
		      "Resource": [
		        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/mysrcenv",
		        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/mydestenv"
		      ]
		    }
		  ]
		}
		```

Action: `TerminateEnvironment`

Resource	Conditions	Example statement

`environment`	`InApplication`
	
	`aws:ResourceTag/` *`key-name`* (Optional)
	
	`aws:TagKeys` (Optional)
		The following policy allows the `TerminateEnvironment` action to terminate the environment **`myenv`** in the application **`My App`**.
		
		JSON
		
		`{`
		`  "Version": "2012-10-17",`
		`  "Statement": [`
		`    {`
		`      "Action": [`
		`        "elasticbeanstalk:TerminateEnvironment"`
		`      ],`
		`      "Effect": "Allow",`
		`      "Resource": [`
		`        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"`
		`      ],`
		`      "Condition": {`
		`        "StringEquals": {`
		`          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"]`
		`        }`
		`      }`
		`    }`
		`  ]`
		`}`

Action: UpdateApplication

Resource	Conditions	Example statement
`application`	`aws:ResourceTag/ key-name` (Optional) `aws:TagKeys` (Optional)	The following policy allows the `UpdateApplication` action to update properties of the application **`My App`**. JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": [ "elasticbeanstalk:UpdateApplication" ], "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App" ] } ] }`

Action: UpdateApplicationResourceLifecycle

Resource	Conditions	Example statement
`application`	`aws:ResourceTag/` *`key-name`* (Optional) `aws:TagKeys` (Optional)	The following policy allows the `UpdateApplicationResourceLifecycle` action to update lifecycle settings of the application **`My App`**. JSON `{ "Version": "2012-10-17", "Statement": [ { "Action": [ "elasticbeanstalk:UpdateApplicationResourceLifecycle" ], "Effect": "Allow", "Resource": [ "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App" ] } ] }`

Action: UpdateApplicationVersion

Resource	Conditions	Example statement
applicati onversion	InApplica tion aws:Resou rceTag/ *key-name* (Optional) aws:TagKeys (Optional)	The following policy allows the UpdateApp licationVersion action to update the propertie s of the application version **My Version** in the application **My App**. JSON `{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:UpdateApplicationVersion"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }`

Action: UpdateConfigurationTemplate

Resource	Conditions	Example statement
`configurationtemplate`	`InApplication` `aws:ResourceTag/ key-name` (Optional) `aws:TagKeys` (Optional)	The following policy allows the `UpdateConfigurationTemplate` action to update the properties or options of the configuration template **`My Template`** in the application **`My App`**. JSON `{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:UpdateConfigurationTemplate"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:configurationtemplate/My App/My Template"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }`

Action: `UpdateEnvironment`

Resource	Conditions	Example statement

`environment`	`InApplication`
	
	`FromApplicationVersion`
	
	`FromConfigurationTemplate`
	
	`aws:ResourceTag/` *`key-name`* (Optional)
	
	`aws:TagKeys` (Optional)
		The following policy allows the `UpdateEnvironment` action to update the environment **`myenv`** in the application **`My App`** by deploying the application version **`My Version`**.
		
		JSON
		
		`{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:UpdateEnvironment"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"], "elasticbeanstalk:FromApplicationVersion": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/My Version"] } } }] }`

Action: `UpdateTagsForResource` – `AddTags`

Resource	Conditions	Example statement

`application`
 	
`applicationversion`
 	
`configurationtemplate`
 	
`environment`
 	
`platform`
	`aws:ResourceTag/` *`key-name`* (Optional)
	
	`aws:RequestTag/` *`key-name`* (Optional)
	
	`aws:TagKeys` (Optional)
		The `AddTags` action is one of two virtual actions associated with the `UpdateTagsForResource` API.
		
		The following policy allows the `AddTags` action to modify tags of existing resources only if they have a tag named `stage` with the value `test`:
		
		JSON
		
		`{`
		`  "Version": "2012-10-17",`
		`  "Statement": [`
		`    {`
		`      "Action": [`
		`        "elasticbeanstalk:AddTags"`
		`      ],`
		`      "Effect": "Allow",`
		`      "Resource": "*",`
		`      "Condition": {`
		`        "StringEquals": {`
		`          "aws:ResourceTag/stage":`
		` ["test"]`
		`        }`
		`      }`
		`    }`
		`  ]`
		`}`

Action: `UpdateTagsForResource` – `RemoveTags`

Resource	Conditions	Example statement

`application`
 	
`applicationversion`
 	
`configurationtemplate`
 	
`environment`
 	
`platform`
	`aws:ResourceTag/` *`key-name`* (Optional)
	
	`aws:TagKeys` (Optional)
		The `RemoveTags` action is one of two virtual actions associated with the [`UpdateTagsForResource`] API.
		
		The following policy denies the `RemoveTags` action to request the removal of a tag named `stage` from existing resources:
		
		JSON
		
		`{ "Version": "2012-10-17", "Statement": [ { "Action": [ "elasticbeanstalk:RemoveTags" ], "Effect": "Deny", "Resource": "*", "Condition": { "ForAnyValue:StringEquals": { "aws:TagKeys": ["stage"] } } } ] }`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:RemoveTags"
      ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:TagKeys": ["stage"]
        }
      }
    }
  ]
}
```

Action: `ValidateConfigurationSettings`

Resource	Conditions	Example statement

`template`
 	
`environment`
	`InApplication`
	
	`aws:ResourceTag/` *`key-name`* (Optional)
	
	`aws:TagKeys` (Optional)
		The following policy allows the `ValidateConfigurationSettings` action to validates configuration settings against the environment **`myenv`** in the application **`My App`**.
		
		JSON
		
		`{ "Version": "2012-10-17", "Statement": [{ "Action": ["elasticbeanstalk:ValidateConfigurationSettings"], "Effect": "Allow", "Resource": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/myenv"], "Condition": { "StringEquals": { "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-east-2:123456789012:application/My App"] } } }] }`

Condition keys for Elastic Beanstalk actions

Keys enable you to specify conditions that express dependencies, restrict permissions, or specify constraints on the input parameters for an action. Elastic Beanstalk supports the following keys.

`InApplication`

Specifies the application that contains the resource that the action operates on.

The following example allows the `UpdateApplicationVersion` action to update the properties of the application version **`My Version`**. The `InApplication` condition specifies **`My App`** as the container for **`My Version`**.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:UpdateApplicationVersion"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My
 App/My Version"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
east-2:123456789012:application/My App"]
        }
      }
    }
  ]
}
```

`FromApplicationVersion`

Specifies an application version as a dependency or a constraint on an input parameter.

The following example allows the `UpdateEnvironment` action to update the environment **`myenv`** in the application **`My App`**. The `FromApplicationVersion` condition constrains the `VersionLabel` parameter to allow only the application version **`My Version`** to update the environment.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:UpdateEnvironment"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/
myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
east-2:123456789012:application/My App"],
          "elasticbeanstalk:FromApplicationVersion":
 ["arn:aws:elasticbeanstalk:us-east-2:123456789012:applicationversion/My App/
My Version"]
        }
      }
    }
  ]
}
```

`FromConfigurationTemplate`

Specifies a configuration template as a dependency or a constraint on an input parameter.

The following example allows the `UpdateEnvironment` action to update the environment **`myenv`** in the application **`My App`**. The `FromConfigurationTemplate` condition constrains the `TemplateName` parameter to allow only the configuration template **`My Template`** to update the environment.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:UpdateEnvironment"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-east-2:123456789012:environment/My App/
myenv"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
east-2:123456789012:application/My App"],
          "elasticbeanstalk:FromConfigurationTemplate":
 ["arn:aws:elasticbeanstalk:us-east-2:123456789012:configurationtemplate/My
 App/My Template"]
        }
      }
    }
  ]
}
```

`FromEnvironment`

Specifies an environment as a dependency or a constraint on an input parameter.

The following example allows the `SwapEnvironmentCNAMEs` action to swap the CNAMEs in **`My App`** for all environments whose names begin with **`mysrcenv`** and **`mydestenv`** but not those environments whose names begin with **`mysrcenvPROD*`** and **`mydestenvPROD*`**.

`FromSolutionStack`

Specifies a solution stack as a dependency or a constraint on an input parameter.

The following policy allows the `CreateConfigurationTemplate` action to create configuration templates whose name begins with **`My Template`** (`My Template*`) in the application **`My App`**. The `FromSolutionStack` condition constrains the `solutionstack` parameter to allow only the solution stack **`32bit Amazon Linux running Tomcat 7`** as the input value for that parameter.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "elasticbeanstalk:CreateConfigurationTemplate"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:elasticbeanstalk:us-
east-2:123456789012:configurationtemplate/My App/My Template*"
      ],
      "Condition": {
        "StringEquals": {
          "elasticbeanstalk:InApplication": ["arn:aws:elasticbeanstalk:us-
east-2:123456789012:application/My App"],
          "elasticbeanstalk:FromSolutionStack": ["arn:aws:elasticbeanstalk:us-
east-2::solutionstack/32bit Amazon Linux running Tomcat 7"]
        }
      }
    }
  ]
}
```

`aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, `aws:TagKeys`

Specify tag-based conditions. For details, see Using tags to control access to Elastic Beanstalk resources.

Using tags to control access to Elastic Beanstalk resources

This topic explains how tag-based access control can help you create and manage IAM policies.

We can use conditions in IAM user policy statements to configure permissions for Elastic Beanstalk's access to resources. To learn more about policy statement conditions, see Resources and conditions for Elastic Beanstalk actions. Using tags in conditions is one way to control access to resources and requests. For information about tagging Elastic Beanstalk resources, see Tagging Elastic Beanstalk application resources.

When you design IAM policies, you might be setting granular permissions by granting access to specific resources. As the number of resources that you manage grows, this task becomes more difficult. Tagging resources and using tags in policy statement conditions can make this task easier. You grant access in bulk to any resource with a certain tag. Then you repeatedly apply this tag to relevant resources, during creation or later.

Tags can be attached to the resource or passed in the request to services that support tagging. In Elastic Beanstalk, resources can have tags, and some actions can include tags. When you create an IAM policy, you can use tag condition keys to control the following conditions:

- Which users can perform actions on an environment, based on tags that it already has.
- What tags can be passed in an action's request.
- Whether specific tag keys can be used in a request.

For the complete syntax and semantics of tag condition keys, see Controlling Access Using Tags in the *IAM User Guide*.

Examples of tag conditions in policies

The following examples demonstrate how to specify tag conditions in policies for Elastic Beanstalk users.

Example 1: Limit actions based on tags in the request

The Elastic Beanstalk **AdministratorAccess-AWSElasticBeanstalk** managed user policy gives users unlimited permission to perform any Elastic Beanstalk action on any Elastic Beanstalk-managed resource.

The following policy limits this power and denies unauthorized users permission to create Elastic Beanstalk production environments. To do that, it denies the `CreateEnvironment` action if the request specifies a tag named `stage` with one of the values `gamma` or `prod`. In addition, the policy prevents these unauthorized users from tampering with the stage of production environments by not allowing tag modification actions to include these same tag values or to completely remove the `stage` tag. A customer's administrator must attach this IAM policy to unauthorized IAM users, in addition to the managed user policy.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "elasticbeanstalk:CreateEnvironment",
        "elasticbeanstalk:AddTags"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/stage": ["gamma", "prod"]
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "elasticbeanstalk:RemoveTags"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:TagKeys": ["stage"]
        }
      }
    }
  ]
}
```

Example 2: Limit actions based on resource tags

The Elastic Beanstalk **AdministratorAccess-AWSElasticBeanstalk** managed user policy gives users unlimited permission to perform any Elastic Beanstalk action on any Elastic Beanstalk-managed resource.

The following policy limits this power and denies unauthorized users permission to perform actions on Elastic Beanstalk production environments. To do that, it denies specific actions if the environment has a tag named `stage` with one of the values `gamma` or `prod`. A customer's administrator must attach this IAM policy to unauthorized IAM users, in addition to the managed user policy.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "elasticbeanstalk:AddTags",
        "elasticbeanstalk:RemoveTags",
        "elasticbeanstalk:DescribeEnvironments",
        "elasticbeanstalk:TerminateEnvironment",
        "elasticbeanstalk:UpdateEnvironment",
        "elasticbeanstalk:ListTagsForResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/stage": ["gamma", "prod"]
        }
      }
    }
  ]
}
```

Example 3: Allow actions based on tags in the request

The following policy grants users permission to create Elastic Beanstalk development applications.

To do that, it allows the `CreateApplication`and `AddTags` actions if the request specifies a tag named `stage` with the value `development`. The `aws:TagKeys` condition ensures that the user can't add other tag keys. In particular, it ensures case sensitivity of the `stage` tag key. Notice that this policy is useful for IAM users that don't have the Elastic Beanstalk **AdministratorAccess-AWSElasticBeanstalk** managed user policy attached. The managed policy gives users unlimited permission to perform any Elastic Beanstalk action on any Elastic Beanstalk-managed resource.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticbeanstalk:CreateApplication",
        "elasticbeanstalk:AddTags"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/stage": "development"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["stage"]
        }
      }
    }
  ]
}
```

Example 4: Allow actions based on resource tags

The following policy grants users permission to perform actions on, and get information about, Elastic Beanstalk development applications.

To do that, it allows specific actions if the application has a tag named `stage` with the value `development`. The `aws:TagKeys` condition ensures that the user can't add other tag keys. In particular, it ensures case sensitivity of the `stage` tag key. Notice that this policy is useful for IAM users that don't have the Elastic Beanstalk **AdministratorAccess-AWSElasticBeanstalk** managed user policy attached. The managed policy gives users unlimited permission to perform any Elastic Beanstalk action on any Elastic Beanstalk-managed resource.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticbeanstalk:UpdateApplication",
        "elasticbeanstalk:DeleteApplication",
        "elasticbeanstalk:DescribeApplications"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/stage": "development"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["stage"]
        }
      }
    }
  ]
}
```

Example policies based on managed policies

This section demonstrates how to control user access to AWS Elastic Beanstalk and includes example policies that provide the required access for common scenarios. These policies are derived from the Elastic Beanstalk managed policies. For information about attaching managed policies to users and groups, see Managing Elastic Beanstalk user policies.

In this scenario, Example Corp. is a software company with three teams responsible for the company website: administrators who manage the infrastructure, developers who build the software for the website, and a QA team that tests the website. To help manage permissions to their Elastic Beanstalk resources, Example Corp. creates three groups to which members of each respective team belong: Admins, Developers, and Testers. Example Corp. wants the Admins group to have full access to all applications, environments, and their underlying resources so that they can create, troubleshoot, and delete all Elastic Beanstalk assets. Developers require permissions to view all Elastic Beanstalk assets and to create and deploy application versions. Developers should not be able to create new applications or environments or terminate running environments. Testers need to view all Elastic Beanstalk resources to monitor and test applications. The Testers should not be able to make changes to any Elastic Beanstalk resources.

The following example policies provide the required permissions for each group.

Example 1: Admins group – All Elastic Beanstalk and related service APIs

The following policy gives users permissions for all actions required to use Elastic Beanstalk. This policy also allows Elastic Beanstalk to provision and manage resources on your behalf in the following services. Elastic Beanstalk relies on these additional services to provision underlying resources when creating an environment.

- Amazon Elastic Compute Cloud

- Elastic Load Balancing
- Auto Scaling
- Amazon CloudWatch
- Amazon Simple Storage Service
- Amazon Simple Notification Service
- Amazon Relational Database Service
- AWS CloudFormation

Note that this policy is an example. It gives a broad set of permissions to the AWS services that Elastic Beanstalk uses to manage applications and environments. For example, `ec2:*` allows an AWS Identity and Access Management (IAM) user to perform any action on any Amazon EC2 resource in the AWS account. These permissions are not limited to the resources that you use with Elastic Beanstalk. As a best practice, you should grant individuals only the permissions they need to perform their duties.

JSON

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "rds:*",
        "cloudformation:*"
      ],
      "Resource" : "*"
    }
  ]
}
```

Example 2: Developers group – All but highly privileged operations

The following policy denies permission to create applications and environments, and allows all other Elastic Beanstalk actions.

Note that this policy is an example. It gives a broad set of permissions to the AWS products that Elastic Beanstalk uses to manage applications and environments. For example, `ec2:*` allows an IAM user to perform any action on any Amazon EC2 resource in the AWS account. These permissions are not limited to the resources that you use with Elastic Beanstalk. As a best practice, you should grant individuals only the permissions they need to perform their duties.

JSON

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Action" : [
        "elasticbeanstalk:CreateApplication",
        "elasticbeanstalk:CreateEnvironment",
        "elasticbeanstalk:DeleteApplication",
        "elasticbeanstalk:RebuildEnvironment",
        "elasticbeanstalk:SwapEnvironmentCNAMEs",
        "elasticbeanstalk:TerminateEnvironment"],
      "Effect" : "Deny",
      "Resource" : "*"
    },
    {
      "Action" : [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "rds:*",
        "cloudformation:*"],
      "Effect" : "Allow",
      "Resource" : "*"
    }
  ]
}
```

Example 3: Testers – View only

The following policy allows read-only access to all applications, application versions, events, and environments. It doesn't allow performing any actions.

JSON

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "elasticbeanstalk:Check*",
        "elasticbeanstalk:Describe*",
        "elasticbeanstalk:List*",
        "elasticbeanstalk:RequestEnvironmentInfo",
        "elasticbeanstalk:RetrieveEnvironmentInfo",
        "ec2:Describe*",
        "elasticloadbalancing:Describe*",
        "autoscaling:Describe*",
        "cloudwatch:Describe*",
        "cloudwatch:List*",
        "cloudwatch:Get*",
        "s3:Get*",
        "s3:List*",
        "sns:Get*",
        "sns:List*",
        "rds:Describe*",
        "cloudformation:Describe*",
        "cloudformation:Get*",
        "cloudformation:List*",
        "cloudformation:Validate*",
        "cloudformation:Estimate*"
      ],
      "Resource" : "*"
    }
  ]
}
```

Example policies based on resource permissions

This section walks through a use case for controlling user permissions for Elastic Beanstalk actions that access specific Elastic Beanstalk resources. We'll walk through the sample policies that support the use case. For more information policies on Elastic Beanstalk resources, see Creating a custom user policy. For information about attaching policies to users and groups, go to Managing IAM Policies in *Using AWS Identity and Access Management*.

In our use case, Example Corp. is a small consulting firm developing applications for two different customers. John is the development manager overseeing the development of the two Elastic Beanstalk applications, app1 and app2. John does development and some testing on the two applications, and only he can update the production environment for the two applications. These are the permissions that he needs for app1 and app2:

- View application, application versions, environments, and configuration templates
- Create application versions and deploy them to the staging environment
- Update the production environment
- Create and terminate environments

Jill is a tester who needs access to view the following resources in order to monitor and test the two applications: applications, application versions, environments, and configuration templates. However, she should not be able to make changes to any Elastic Beanstalk resources.

Jack is the developer for app1 who needs access to view all resources for app1 and also needs to create application versions for app1 and deploy them to the staging environment.

Judy is the administrator of the AWS account for Example Corp. She has created IAM users for John, Jill, and Jack and attaches the following policies to those users to grant the appropriate permissions to the app1 and app2 applications.

Example 1: John – Development manager for app1, app2

We have broken down John's policy into three separate policies so that they are easier to read and manage. Together, they give John the permissions he needs to perform development, testing, and deployment actions on the two applications.

The first policy specifies actions for Auto Scaling, Amazon S3, Amazon EC2, CloudWatch, Amazon SNS, Elastic Load Balancing, Amazon RDS, and AWS CloudFormation. Elastic Beanstalk relies on these additional services to provision underlying resources when creating an environment.

Note that this policy is an example. It gives a broad set of permissions to the AWS products that Elastic Beanstalk uses to manage applications and environments. For example, `ec2:*` allows an IAM user to perform any action on any Amazon EC2 resource in the AWS account. These permissions are not limited to the resources that you use with Elastic Beanstalk. As a best practice, you should grant individuals only the permissions they need to perform their duties.

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [
   {
   "Effect": "Allow",
   "Action": [
   "ec2:*",
   "ecs:*",
   "ecr:*",
   "elasticloadbalancing:*",
   "autoscaling:*",
   "cloudwatch:*",
   "s3:*",
   "sns:*",
   "cloudformation:*",
   "dynamodb:*",
   "rds:*",
   "sqs:*",
   "logs:*",
   "iam:GetPolicyVersion",
   "iam:GetRole",
   "iam:ListRolePolicies",
   "iam:ListAttachedRolePolicies",
   "iam:ListInstanceProfiles",
   "iam:ListRoles",
   "iam:ListServerCertificates",
   "acm:DescribeCertificate",
   "acm:ListCertificates",
   "codebuild:CreateProject",
   "codebuild:DeleteProject",
   "codebuild:BatchGetBuilds",
   "codebuild:StartBuild"
   ],
   "Resource": "*"
   },
   {
   "Effect": "Allow",
   "Action": [
   "iam:PassRole"
   ],
   "Resource": "arn:aws:iam::111122223333:role/MyRole"
   }
   ]
   }
```

The second policy specifies the Elastic Beanstalk actions that John is allowed to perform on the app1 and app2 resources. The `AllCallsInApplications` statement allows all Elastic Beanstalk actions (`"elasticbeanstalk:*"`) performed on all resources within app1 and app2 (for example, `elasticbeanstalk:CreateEnvironment`). The `AllCallsOnApplications` statement allows all Elastic Beanstalk actions (`"elasticbeanstalk:*"`) on the app1 and app2 application resources (for example, `elasticbeanstalk:DescribeApplications`, `elasticbeanstalk:UpdateApplication`, etc.). The `AllCallsOnSolutionStacks` statement allows all Elastic Beanstalk actions (`"elasticbeanstalk:*"`) for solution stack resources (for example, `elasticbeanstalk:ListAvailableSolutionStacks`).

JSON

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Sid":"AllCallsInApplications",
         "Action":[
            "elasticbeanstalk:*"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ],
         "Condition":{
            "StringEquals":{
               "elasticbeanstalk:InApplication":[
                  "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/
app1",
                  "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/
app2"
               ]
            }
         }
      },
      {
         "Sid":"AllCallsOnApplications",
         "Action":[
            "elasticbeanstalk:*"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app1",
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app2"
         ]
      },
      {
         "Sid":"AllCallsOnSolutionStacks",
         "Action":[
            "elasticbeanstalk:*"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:elasticbeanstalk:us-east-2::solutionstack/*"
         ]
      }
   ]
}
```

The third policy specifies the Elastic Beanstalk actions that the second policy needs permissions to in order to complete those Elastic Beanstalk actions. The `AllNonResourceCalls` statement allows the `elasticbeanstalk:CheckDNSAvailability` action, which is required to call `elasticbeanstalk:CreateEnvironment` and other actions. It also allows the `elasticbeanstalk:CreateStorageLocation` action, which is required for `elasticbeanstalk:CreateApplication`, `elasticbeanstalk:CreateEnvironment`, and other actions.

JSON

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Sid":"AllNonResourceCalls",
         "Action":[
            "elasticbeanstalk:CheckDNSAvailability",
            "elasticbeanstalk:CreateStorageLocation"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ]
      }
   ]
}
```

Example 2: Jill – Tester for app1, app2

We have broken down Jill's policy into three separate policies so that they are easier to read and manage. Together, they give Jill the permissions she needs to perform testing and monitoring actions on the two applications.

The first policy specifies `Describe*`, `List*`, and `Get*` actions on Auto Scaling, Amazon S3, Amazon EC2, CloudWatch, Amazon SNS, Elastic Load Balancing, Amazon RDS, and AWS CloudFormation (for non-legacy container types) so that the Elastic Beanstalk actions are able to retrieve the relevant information about the underlying resources of the app1 and app2 applications.

JSON

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "ec2:Describe*",
            "elasticloadbalancing:Describe*",
            "autoscaling:Describe*",
            "cloudwatch:Describe*",
            "cloudwatch:List*",
            "cloudwatch:Get*",
            "s3:Get*",
            "s3:List*",
            "sns:Get*",
            "sns:List*",
            "rds:Describe*",
            "cloudformation:Describe*",
         "cloudformation:Get*",
         "cloudformation:List*",
         "cloudformation:Validate*",
         "cloudformation:Estimate*"
         ],
         "Resource":"*"
      }
   ]
}
```

The second policy specifies the Elastic Beanstalk actions that Jill is allowed to perform on the app1 and app2 resources. The `AllReadCallsInApplications` statement allows her to call the `Describe*` actions and the environment info actions. The `AllReadCallsOnApplications` statement allows her to call the `DescribeApplications` and `DescribeEvents` actions on the app1 and app2 application resources. The `AllReadCallsOnSolutionStacks` statement allows viewing actions that involve solution stack resources (`ListAvailableSolutionStacks`, `DescribeConfigurationOptions`, and `ValidateConfigurationSettings`).

JSON

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Sid":"AllReadCallsInApplications",
         "Action":[
            "elasticbeanstalk:Describe*",
            "elasticbeanstalk:RequestEnvironmentInfo",
            "elasticbeanstalk:RetrieveEnvironmentInfo"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ],
         "Condition":{
            "StringEquals":{
               "elasticbeanstalk:InApplication":[
                  "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/
app1",
                  "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/
app2"
               ]
            }
         }
      },
      {
         "Sid":"AllReadCallsOnApplications",
         "Action":[
            "elasticbeanstalk:DescribeApplications",
            "elasticbeanstalk:DescribeEvents"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app1",
            "arn:aws:elasticbeanstalk:us-east-2:123456789012:application/app2"
         ]
      },
      {
         "Sid":"AllReadCallsOnSolutionStacks",
         "Action":[
            "elasticbeanstalk:ListAvailableSolutionStacks",
            "elasticbeanstalk:DescribeConfigurationOptions",
            "elasticbeanstalk:ValidateConfigurationSettings"
         ],
         "Effect":"Allow",
         "Resource":[
            "arn:aws:elasticbeanstalk:us-east-2::solutionstack/*"
         ]
      }
   ]
}
```

The third policy specifies the Elastic Beanstalk actions that the second policy needs permissions to in order to complete those Elastic Beanstalk actions. The `AllNonResourceCalls` statement allows the `elasticbeanstalk:CheckDNSAvailability` action, which is required for some viewing actions.

JSON

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Sid":"AllNonResourceCalls",
         "Action":[
            "elasticbeanstalk:CheckDNSAvailability"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ]
      }
   ]
}
```

Example 3: Jack – Developer for app1

We have broken down Jack's policy into three separate policies so that they are easier to read and manage. Together, they give Jack the permissions he needs to perform testing, monitoring, and deployment actions on the app1 resource.

The first policy specifies the actions on Auto Scaling, Amazon S3, Amazon EC2, CloudWatch, Amazon SNS, Elastic Load Balancing, Amazon RDS, and AWS CloudFormation (for non-legacy container types) so that the Elastic Beanstalk actions are able to view and work with the underlying resources of app1. For a list of supported non-legacy container types, see the section called “Why are some platform versions marked legacy?”

Note that this policy is an example. It gives a broad set of permissions to the AWS products that Elastic Beanstalk uses to manage applications and environments. For example, `ec2:*` allows an IAM user to perform any action on any Amazon EC2 resource in the AWS account. These permissions are not limited to the resources that you use with Elastic Beanstalk. As a best practice, you should grant individuals only the permissions they need to perform their duties.

JSON

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "ec2:*",
            "elasticloadbalancing:*",
            "autoscaling:*",
            "cloudwatch:*",
            "s3:*",
            "sns:*",
            "rds:*",
            "cloudformation:*"
         ],
         "Resource":"*"
      }
   ]
}
```

The second policy specifies the Elastic Beanstalk actions that Jack is allowed to perform on the app1 resource.

The third policy specifies the Elastic Beanstalk actions that the second policy needs permissions to in order to complete those Elastic Beanstalk actions. The `AllNonResourceCalls` statement allows the `elasticbeanstalk:CheckDNSAvailability` action, which is required to call `elasticbeanstalk:CreateEnvironment` and other actions. It also allows the `elasticbeanstalk:CreateStorageLocation` action, which is required for `elasticbeanstalk:CreateEnvironment`, and other actions.

JSON

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
         "Sid":"AllNonResourceCalls",
         "Action":[
            "elasticbeanstalk:CheckDNSAvailability",
            "elasticbeanstalk:CreateStorageLocation"
         ],
         "Effect":"Allow",
         "Resource":[
            "*"
         ]
      }
   ]
}
```

Preventing cross-environment Amazon S3 bucket access

This topic explains how managed policies may allow cross-environment S3 bucket access and how you can create custom policies to manage this type of access.

Elastic Beanstalk provides managed polices to handle the AWS resources required by the Elastic Beanstalk environments in your AWS account. The permissions provided by default to one application in your AWS account have access to S3 resources that belong to other applications in the same AWS account.

If your AWS account runs multiple Beanstalk applications, you can scope down the security of your policies by creating your own custom policy to attach to your own service role or instance profile for each environment. You can then limit the S3 permissions in your custom policy to a specific environment.

> **Note**
>
> Be aware that you're responsible for maintaining your custom policy. If an Elastic Beanstalk managed policy on which your custom policy is based changes, you'll need to modify your custom policy with the respective changes to the base policy. For a change history of Elastic Beanstalk managed policies, see Elastic Beanstalk updates to AWS managed policies.

Example of scoped down permissions

The following example is based on the AWSElasticBeanstalkWebTier managed policy.

The default policy includes the following lines for permissions to S3 buckets. This default policy doesn't limit the S3 bucket actions to specific environments or applications.

```
{
   "Sid" : "BucketAccess",
   "Action" : [
      "s3:Get*",
      "s3:List*",
      "s3:PutObject"
     ],
   "Effect" : "Allow",
   "Resource" : [
      "arn:aws:s3:::elasticbeanstalk-*",
      "arn:aws:s3:::elasticbeanstalk-*/*"
     ]
}
```

You can scope down the access by qualifying specific resources to a service role specified as a `Principal`. The following example provides the custom service role `aws-elasticbeanstalk-ec2-role-my-example-env` permissions to S3 buckets in the environment with id `my-example-env-ID`.

Example Grant permissions to only a specific environment's S3 buckets

```
{
   "Sid": "BucketAccess",
   "Action": [
      "s3:Get*",
      "s3:List*",
      "s3:PutObject"
    ],
   "Effect": "Allow",
   "Principal": {
      "AWS": "arn:aws:iam::...:role/aws-elasticbeanstalk-ec2-role-my-example-env"
     },
   "Resource": [
      "arn:aws:s3:::elasticbeanstalk-my-region-account-id-12345",
      "arn:aws:s3:::elasticbeanstalk-my-region-account-id-12345/resources/environments/
my-example-env-ID/*"
```

```
    ]
}
```

Note

The Resource ARN must include the Elastic Beanstalk environment ID, (not the environment name). You can obtain the environment id from the Elastic Beanstalk console on the Environment overview page. You can also use the AWS CLI describe-environments command to obtain this information.

For more information to help you update S3 bucket permissions for your Elastic Beanstalk environments, see the following resources:

- Using Elastic Beanstalk with Amazon S3 in this guide
- Resource types defined by Amazon S3 in the *Service Authorization Reference* guide
- ARN format in the *IAM User Guide*

Using Elastic Beanstalk with Amazon RDS

This section explains how you can use Elastic Beanstalk with Amazon Relational Database Service (Amazon RDS) to set up, operate, and scale a relational database. We explain some concepts about configuration and provide recommendations. Then we'll walk you through the process to create and connect to an Amazon RDS.

There are two options to get started:

- Create a new database in Amazon RDS.
- Start with a database that was previously created by Elastic Beanstalk and subsequently decoupled from a Beanstalk environment. For more information, see the section called "Database".

Select approach

You can use either approach to run a database instance in Amazon RDS and configure your application to connect to it on launch. You can connect multiple environments to a database.

Note

If you haven't used a database instance with your application before, we recommend that you add a database to a test environment with the Elastic Beanstalk console first. By doing this, you can verify that your application can read the environment properties, construct a connection string, and connect to a database instance, without the additional configuration work required for a standalone database. For more information, see Adding a database to your Elastic Beanstalk environment.

Configure a security group

To allow the Amazon EC2 instances in your environment to connect to an outside database, configure an additional security group for the Auto Scaling group that's associated with your environment. You can attach the same security group that's attached to your database instance. Or, you can use a separate security group. If you attach a different security group, you must configure the security group that's attached to your database to allow inbound access from this security group.

Note

You can connect your environment to a database by adding a rule to the security group that's attached to your database. This rule must allow inbound access from the autogenerated security group that Elastic Beanstalk attaches to the Auto Scaling group for your environment. However, know that, by creating this rule, you also create a dependency between the two security groups. Subsequently, when you attempt to terminate the environment, Elastic Beanstalk will be unable to delete the environment's security group, because the database's security group is dependent on it.

Configure the database connection

After you launch your database instance and configure security groups, you can pass the connection information, such as the endpoint and password, to your application by using environment properties. This is the same mechanism that Elastic Beanstalk uses in the background when you run a database instance in your environment.

For an additional layer of security, you can store your connection information in Amazon S3, and configure Elastic Beanstalk to retrieve it during deployment. With configuration files

(`.ebextensions`), you can configure the instances in your environment to securely retrieve files from Amazon S3 when you deploy your application.

Topics

Launching and connecting to an external Amazon RDS instance in a default VPC

The following procedures describe the process for connecting to an external Amazon RDS instance to a default VPC. The process is the same if you're using a custom VPC. The only additional requirements are that your environment and DB instance are in the same subnet, or in subnets that are allowed to communicate with each other. For more information about configuring a custom VPC to use with Elastic Beanstalk, see Using Elastic Beanstalk with Amazon VPC.

Note

- An alternative to launching a new DB instance, is to start with a database that was previously created by Elastic Beanstalk and subsequently decoupled from a Beanstalk environment. For more information, see the section called “Database”. With this option, you don't need to complete the procedure for launching a new database. However, you do need to complete the subsequent procedures that are described in this topic.
- If you’re starting with a database that was created by Elastic Beanstalk and subsequently decoupled from a Beanstalk environment, you can skip the first group of steps and continue with the steps grouped under *To modify the inbound rules on your RDS instance's security group*.
- If you plan to use the database that you decouple for a production environment, verify the storage type that the database uses is suitable for your workload. For more information, see DB Instance Storage and Modifying a DB instance in the *Amazon RDS User Guide*.

To launch an RDS DB instance in a default VPC

1. Open the RDS console.
2. In the navigation pane, choose **Databases**.
3. Choose **Create database**.
4. Choose **Standard Create**.

   > **Important**
   >
   > Do not choose **Easy Create**. If you choose it, you can't configure the necessary settings to launch this RDS DB.

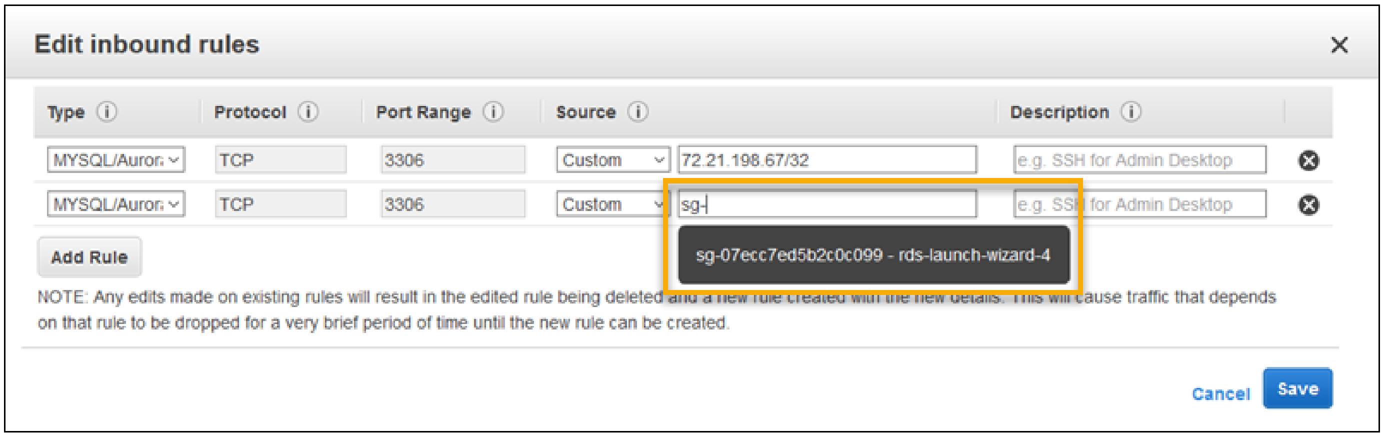
5. Under **Additional configuration**, for **Initial database name**, type **ebdb**.
6. Review the default settings and adjust these settings according to your specific requirements. Pay attention to the following options:
   - **DB instance class** – Choose an instance size that has an appropriate amount of memory and CPU power for your workload.
   - **Multi-AZ deployment** – For high availability, set this to **Create an Aurora Replica/Reader node in a different AZ**.
   - **Master username** and **Master password** – The database username and password. Make a note of these settings because you will use them later.
7. Verify the default settings for the remaining options, and then choose **Create database**.

Next, modify the security group that's attached to your DB instance to allow inbound traffic on the appropriate port. This is the same security group that you will attach to your Elastic Beanstalk environment later. As a result, the rule that you add will grant inbound access permission to other resources in the same security group.

To modify the inbound rules on the security group that's attached to your RDS instance

1. Open the Amazon RDS console.
2. Choose **Databases**.
3. Choose the name of your DB instance to view its details.
4. In the **Connectivity** section, make a note of the **Subnets**, **Security groups**, and **Endpoint** that are displayed on this page. This is so you can use this information later.

5. Under **Security**, you can see the security group that's associated with the DB instance. Open the link to view the security group in the Amazon EC2 console.
6. In the security group details, choose **Inbound**.
7. Choose **Edit**.
8. Choose **Add Rule**.
9. For **Type**, choose the DB engine that your application uses.
10. For **Source**, type **sg-** to view a list of available security groups. Choose the security group that's associated with the Auto Scaling group that's used with your Elastic Beanstalk environment. This is so that Amazon EC2 instances in the environment can have access to the database.


Edit inbound rules

Type	Protocol	Port Range	Source		Description
MYSQL/Auror	TCP	3306	Custom	72.21.198.67/32	e.g. SSH for Admin Desktop
MYSQL/Auror	TCP	3306	Custom	sg-	e.g. SSH for Admin Desktop

sg-07ecc7ed5b2c0c099 - rds-launch-wizard-4

NOTE: Any edits made on existing rules will result in the edited rule being deleted and a new rule created with the new details. This will cause traffic that depends on that rule to be dropped for a very brief period of time until the new rule can be created.

11. Choose **Save**.

Next, add the security group for the DB instance to your running environment. In this procedure Elastic Beanstalk re-provisions all instances in your environment with the additional security group attached.

To add a security group to your environment

- Do one of the following:
  - To add a security group using the Elastic Beanstalk console

    a. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
    b. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
    c. In the navigation pane, choose **Configuration**.
    d. In the **Instances** configuration category, choose **Edit**.

   e. Under **EC2 security groups**, choose the security group to attach to the instances, in addition to the instance security group that Elastic Beanstalk creates.

   f. To save the changes choose **Apply** at the bottom of the page.

   g. Read the warning, and then choose **Confirm**.

- To add a security group using a configuration file, use the `securitygroup-addexisting.config` example file.

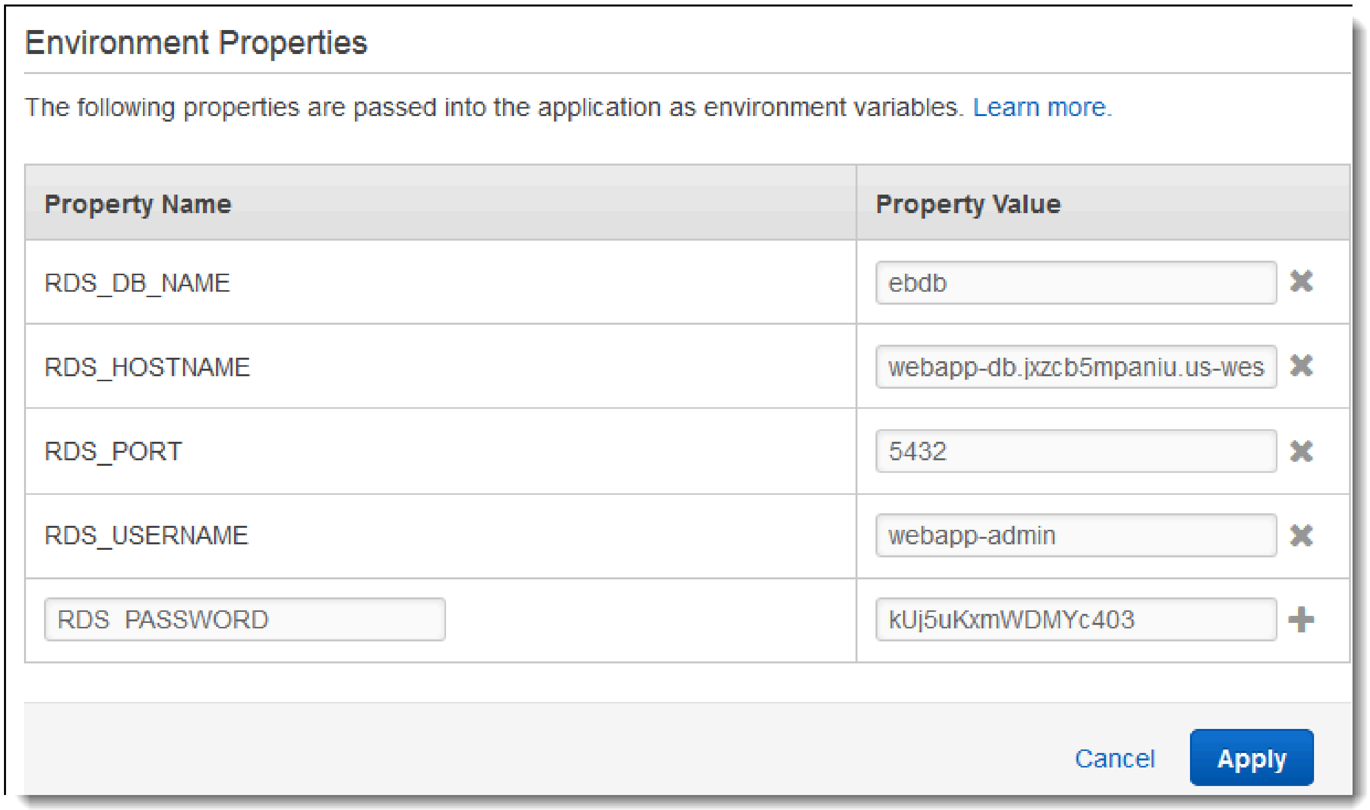
Next, pass the connection information to your environment by using environment properties. When you add a DB instance to your environment with the Elastic Beanstalk console, Elastic Beanstalk uses environment properties, such as **RDS_HOSTNAME**, to pass connection information to your application. You can use the same properties. By doing this, you use the same application code with both integrated DB instances and external DB instances. Or, alternatively, you can choose your own property names.

To configure environment properties for an Amazon RDS DB instance

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. In the **Environment properties** section, define the variables that your application reads to construct a connection string. For compatibility with environments that have an integrated RDS DB instance, use the following names and values. You can find all values, except for your password, in the RDS console.

Property name	Description	Property value
`RDS_HOSTNAME`	The hostname of the DB instance.	On the **Connectivity & security** tab on the Amazon RDS console: **Endpoint**.
`RDS_PORT`	The port where the DB instance accepts connectio	On the **Connectivity & security** tab on the Amazon RDS console: **Port**.

Property name	Description	Property value
	ns. The default value varies among DB engines.	
`RDS_DB_NAME`	The database name, **ebdb**.	On the **Configuration** tab on the Amazon RDS console: **DB Name**.
`RDS_USERNAME`	The username that you configured for your database.	On the **Configuration** tab on the Amazon RDS console: **Master username**.
`RDS_PASSWORD`	The password that you configured for your database.	Not available for reference in the Amazon RDS console.



Environment Properties

The following properties are passed into the application as environment variables. Learn more.

Property Name	Property Value
RDS_DB_NAME	ebdb
RDS_HOSTNAME	webapp-db.jxzcb5mpaniu.us-wes
RDS_PORT	5432
RDS_USERNAME	webapp-admin
RDS_PASSWORD	kUj5uKxmWDMYc403

6. To save the changes choose **Apply** at the bottom of the page.

If you didn't program your application to read environment properties and construct a connection string yet, see the following language-specific topics for instructions:

- Java SE – Connecting to a database (Java SE platforms)
- Java with Tomcat – Connecting to a database (Tomcat platforms)
- Node.js – Connecting to a database
- .NET – Connecting to a database
- PHP – Connecting to a database with a PDO or MySQLi
- Python – Connecting to a database
- Ruby – Connecting to a database

Finally, depending on when your application reads environment variables, you might need to restart the application server on the instances in your environment.

To restart your environment's app servers

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. Choose **Actions**, and then choose **Restart app server(s)**.

Storing the Amazon RDS credentials in AWS Secrets Manager

This topic explains how AWS Secrets Manager can improve your security posture for credential retrieval with Elastic Beanstalk. It also provides references to specific resources that can help you configure credentials for your Elastic Beanstalk application.

AWS Secrets Manager helps you improve your security posture, by providing the ability to store and retrieve encrypted credentials. Storing the credentials in Secrets Manager helps avoid possible compromise by anyone who can inspect your application or the components related to it. Your code can make a runtime call to the Secrets Manager service to retrieve credentials dynamically. Secrets Manager also offers features like client-side secret caching components for runtime languages, which include Python, Go, and Java.

For more information, see the following topics in the *AWS Secrets Manager User Guide*.

- How Amazon RDS uses AWS Secrets Manager
- Create an AWS Secrets Manager database secret
- Retrieve secrets from AWS Secrets Manager

Cleaning up an external Amazon RDS instance

When you connect an external Amazon RDS instance to your Elastic Beanstalk environment, the database instance isn't dependent upon your environment's lifecycle, and, therefore, it isn't deleted when you terminate your environment. To ensure that personal information that you might have stored in the database instance isn't unnecessarily retained, delete any records that you don't need anymore. Alternatively, delete the database instance.

Using Elastic Beanstalk with Amazon S3

This topic explains how Elastic Beanstalk utilizes Amazon Simple Storage Service (Amazon S3) and the types of objects that it stores in S3 buckets. It also notes which objects you must delete manually after you terminate your Elastic Beanstalk environment and provides instructions to do so.

The Elastic Beanstalk Amazon S3 customer account bucket

Elastic Beanstalk creates an encrypted Amazon S3 bucket named `elasticbeanstalk-region-account-id` for each region in which you create environments. Your AWS account owns this bucket. Elastic Beanstalk stores temporary configuration files and other objects for the proper operation of your application in this bucket. Elastic Beanstalk requires enabled ACLs for service-managed buckets and therefore enables this bucket's Access Control List (ACL).

Be aware that Amazon S3 disables bucket Access Control Lists (ACLs) by default. Furthermore, the ACL overview topic in the *Amazon S3 User Guide* recommends that you keep ACLs disabled, except for specific use cases. The Elastic Beanstalk service-managed buckets fall into a use case that requires enabled ACLs. To maintain security Elastic Beanstalk deployments enforce that this bucket is owned by the account running the application.

Elastic Beanstalk retains the default encryption provided by Amazon S3 buckets. For more information about bucket encryption, see Amazon S3 default encryption in the *Amazon Simple Storage Service User Guide*.

Contents of the Elastic Beanstalk Amazon S3 customer account bucket

The following table lists some objects that Elastic Beanstalk stores in your customer account bucket. The table also shows which objects have to be deleted manually. To avoid unnecessary storage costs, and to ensure that personal information isn't retained, be sure to manually delete these objects when you no longer need them.

Object	When stored?	When deleted?
Application versions	When you create an environment or deploy your application code to an existing environment, Elastic Beanstalk stores an application version in Amazon S3 and associates it with the environment.	During application deletion, and according to Version lifecycle.
Source bundles	When you upload a new application version using the Elastic Beanstalk console or the EB CLI, Elastic Beanstalk stores a copy of it in Amazon S3, and sets it as your environment's source bundle.	*Manually.* When you delete an application version, you can choose **Delete versions from Amazon S3** to also delete the related source bundle. For details, see Managing application versions.
Custom platforms	When you create a custom platform, Elastic Beanstalk temporarily stores related data in Amazon S3.	Upon successful completion of the custom platform's creation.
Log files	You can request Elastic Beanstalk to retrieve instance log files (tail or bundle logs) and store them in Amazon S3. You can also enable log rotation and configure your environment to publish logs automatically to Amazon S3 after they are rotated.	Tail and bundle logs: 15 minutes after they are created. Rotated logs: *Manually.*
Saved configurations	*Manually.*	*Manually.*

Deleting objects in the Elastic Beanstalk Amazon S3 bucket

When you terminate an environment or delete an application, Elastic Beanstalk deletes most related objects from Amazon S3. To minimize storage costs of a running application, routinely delete objects that your application doesn't need. In addition, pay attention to objects that you have to delete manually, as listed in Contents of the Elastic Beanstalk Amazon S3 customer account bucket. To ensure that private information isn't unnecessarily retained, delete these objects when you don't need them anymore.

- Delete application versions that you don't expect to use in your application anymore. When you delete an application version, you can select **Delete versions from Amazon S3** to also delete the related source bundle—a copy of your application's source code and configurations files, which Elastic Beanstalk uploaded to Amazon S3 when you deployed an application or uploaded an application version. To learn how to delete an application version, see Managing application versions.
- Delete rotated logs that you don't need. Alternatively, download them or move them to Amazon S3 Glacier for further analysis.
- Delete saved configurations that you aren't going to use in any environment anymore.

Deleting the Elastic Beanstalk Amazon S3 bucket

When Elastic Beanstalk creates a bucket it also creates a bucket policy that it applies to the new bucket. This policy servers two purposes:

- To allow environments to write to the bucket.
- To prevent accidental deletion of the bucket.

Due to the policy that Elastic Beanstalk applies to the buckets that it creates for your environments, you're not be allowed to delete these buckets, unless you deliberately delete the bucket policy first. You can delete the bucket policy from the **Permissions** section of the bucket properties in the Amazon S3 console.

Warning

We recommend that you delete specific unnecessary objects from your Elastic Beanstalk Amazon S3 bucket, instead of deleting the entire bucket.

If you delete a bucket that Elastic Beanstalk created in your account, and you still have existing applications and running environments in the corresponding region, your applications might stop working correctly. For example:

- When an environment scales out, Elastic Beanstalk should be able to find the environment's application version in the Amazon S3 bucket and use it to start new Amazon EC2 instances.
- When you create a custom platform, Elastic Beanstalk uses temporary Amazon S3 storage during the creation process.

For more information about the implications of deleting an S3 bucket, see the considerations listed in Deleting a bucket in the *Amazon S3 User Guide*.

To delete an Elastic Beanstalk storage bucket (console)

The general procedure to delete an S3 bucket is also described in Deleting a bucket in the *Amazon S3 User Guide*. Since we're deleting a bucket created by Elastic Beanstalk in the following procedure, we include additional steps to delete the bucket policy first.

1. Open the Amazon S3 console.
2. Open the Elastic Beanstalk storage bucket's page by choosing the bucket name.
3. Choose the **Permissions** tab.
4. Choose **Bucket Policy**.
5. Choose **Delete**.
6. Go back to the Amazon S3 console's main page, and then select the Elastic Beanstalk storage bucket.
7. Choose **Delete Bucket**.
8. Confirm that you want to delete the bucket by entering the bucket name into the text field, and then choose **Delete bucket**.

Using Elastic Beanstalk with AWS Secrets Manager and AWS Systems Manager Parameter Store

This topic explains how you can use AWS Secrets Manager and AWS Systems Manager Parameter Store with your Elastic Beanstalk environment to securely store and retrieve sensitive information, such as credentials and API keys. Your application can retrieve stored secrets and parameters directly from these stores, using the APIs or command line tools of these services.

Elastic Beanstalk also offers the ability to reference Secrets Manager and Systems Manager Parameter Store data in environment variables. This is a secure option for your application to natively access secrets and parameters stored by these services without having to manage API calls to them.

Topics

Fetching secrets and parameters to Elastic Beanstalk environment variables

Elastic Beanstalk can fetch values from AWS Secrets Manager and AWS Systems Manager Parameter Store during instance bootstrapping and assign them to environment variables for your application to use.

The following points summarize configuration, synchronization and access for using environment variables as secrets:

- Configure your environment variables to store secrets by specifying the Amazon Resource Names (ARNs) for the secrets and parameters they will store.
- When secret values are updated or rotated in Secrets Manager or Systems Manager Parameter Store, you must manually refresh your environment variables.
- The secrets environment variables are available to ebextension container commands and platform hooks.

Supported platform versions

Platform versions that were released on or after March 26, 2025 support AWS Secrets Manager secrets and AWS Systems Manager Parameter Store parameters configured as environment variables.

Note

With the exception of the Docker and ECS based docker platforms, the Amazon Linux 2 platform versions don't support multiline variable values. For more information about multiline variable support, see Multiline values.

Topics

Pricing

Standard charges apply for using Secrets Manager and Systems Manager Parameter Store. For more information about pricing, see the following websites:

- AWS Secrets Manager pricing
- AWS Systems Manager pricing (select *Parameter Store* from the content list)

Elastic Beanstalk doesn't charge for your application to reference environment secrets via environment variables. However, standard charges do apply to requests that Elastic Beanstalk makes to these services on your behalf.

Configure secrets as Elastic Beanstalk environment variables

You can use the Elastic Beanstalk console, configuration files in `.ebextensions`, the AWS CLI, and the AWS SDK to configure secrets and parameters as environment variables.

Topics

Prerequisites

Before you can set up your environment variables to reference secrets you'll first need to complete the following steps.

General procedure prior to environment variable configuration

1. Create the Secrets Manager secrets or the Parameter Store parameters to store your sensitive data. For more information, see one or both of the following topics:
   - *Creating secrets* in the section called “Using Secrets Manager”
   - *Creating parameters* in the section called “Using Systems Manager Parameter Store”
2. Set up the required IAM permissions for your environment’s EC2 instances to fetch the secrets and parameters. For more information, see Required IAM permissions.

Using the console

You can use the Elastic Beanstalk console to configure secrets as environment variables.

To configure secrets as environment variables in the Elastic Beanstalk console

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.
3. In the navigation pane, choose **Configuration**.
4. In the **Updates, monitoring, and logging** configuration category, choose **Edit**.
5. Scroll down to **Runtime environment variables**.
6. Select **Add environment variable**.
7. For **Source** select either **Secrets Manager** or **SSM Parameter Store**.

> **Note**
>
> For more information about the **Plain text** option in the drop-down, see Configuring environment properties (environment variables).

8. For **Environment variable name** enter the name of the environment variable to hold the secret or parameter value.
9. For **Environment variable value** enter the ARN of the Systems Manager Parameter Store parameter or the Secrets Manager secret. During instance bootstrapping Elastic Beanstalk will initiate the value of the variable you entered in **Step 8** to the value stored in this ARN resource.

   The console validates if the value you enter is a valid ARN format for the store that you select in **Step 7**. However, it does not validate the existence of the resource specified by the ARN or if you have the required IAM permissions to access to it.
10. If you need to add more variables repeat **Step 6** through **Step 9**.
11. To save the changes choose **Apply** at the bottom of the page.

Configuration using files in .ebextensions

You can use Elastic Beanstalk configuration files to configure secrets as environment variables. Use the aws:elasticbeanstalk:application:environmentsecrets namespace to define environment properties.

Example .ebextensions/options.config for environment secrets (shorthand syntax)

```
option_settings:
  aws:elasticbeanstalk:application:environmentsecrets:
    MY_SECRET: arn:aws:secretsmanager:us-east-1:111122223333:secret:mysecret
    MY_PARAMETER: arn:aws:ssm:us-east-1:111122223333:parameter/myparam
```

Example .ebextensions/options.config for environment secrets (standard syntax)

```
option_settings:
  - namespace: aws:elasticbeanstalk:application:environmentsecrets
    option_name: MY_SECRET
    value: arn:aws:secretsmanager:us-east-1:111122223333:secret:mysecret
  - namespace: aws:elasticbeanstalk:application:environmentsecrets
```

```
    option_name: MY_PARAMETER
    value: arn:aws:ssm:us-east-1:111122223333:parameter/myparam
```

Configuration using the AWS CLI

You can use the AWS Command Line Interface (AWS CLI) to configure secrets as Elastic Beanstalk environment variables. This section provides examples of the create-environment and update-environment commands with the aws:elasticbeanstalk:application:environmentsecrets namespace. When Elastic Beanstalk bootstraps the EC2 instances for the environments that these command reference, it initializes the environment variables with the fetched secret and the parameter values. It fetches these values from the respective ARNs of Secrets Manager and Systems Manager Parameter Store.

The two following examples use the create-environment command to add a secret and a parameter, configured as environment variables named `MY_SECRET`and `MY_PARAMETER`.

Example of create-environment with secrets configured as environment variables (namespace options inline)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v6.5.0 running Node.js 20" \
--option-settings \
Namespace=aws:elasticbeanstalk:application:environmentsecrets,OptionName=MY_SECRET,Value=arn:aw
east-1:111122223333:secret:mysecret \
Namespace=aws:elasticbeanstalk:application:environmentsecrets,OptionName=MY_PARAMETER,Value=arn
east-1:111122223333:parameter/myparam
```

As an alternative, use an `options.json` file to specify the namespace options instead of including them inline.

Example of create-environment with secrets configured as environment variables (namespace options in `options.json` file)

```
aws elasticbeanstalk create-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v6.5.0 running Node.js 20" \
--option-settings file://options.json
```

Example

```
### example options.json ###
[
  {
    "Namespace": "aws:elasticbeanstalk:application:environmentsecrets",
    "OptionName": "MY_SECRET",
    "Value": "arn:aws:secretsmanager:us-east-1:111122223333:secret:mysecret"
  },
  {
    "Namespace": "aws:elasticbeanstalk:application:environmentsecrets",
    "OptionName": "MY_PARAMETER",
    "Value": "arn:aws:ssm:us-east-1:111122223333:parameter/myparam"
  }
]
```

The next example configures environment variables, named `MY_SECRET`and `MY_PARAMETER`, to store a secret and a parameter for an existing environment. The update-environment command passes options with the same syntax as the `create-environment` command, either inline or with an `options.json` file. The following example demonstrates the command using the same `options.json` file that's also used in the previous example.

Example of update-environment with secrets configured as environment variables (namespace options in `options.json` file)

```
aws elasticbeanstalk update-environment \
--region us-east-1 \
--application-name my-app \
--environment-name my-env \
--solution-stack-name "64bit Amazon Linux 2023 v6.5.0 running Node.js 20" \
--option-settings file://options.json
```

Configuration using the AWS SDK

You can configure secrets and parameters as environment variables using the AWS SDKs. Similar to the `update-environment` and `create-environment` AWS CLI commands mentioned in the previous section, you can use the CreateEnvironment and UpdateEnvironment API actions. Use the `OptionSettings` request parameter to specify the options of the aws:elasticbeanstalk:application:environmentsecrets namespace.

Best practices for secrets synchronization with Elastic Beanstalk environment variables

This topic recommends best practices for your application to use environment secrets with Secrets Manager or the Systems Manager Parameter Store. Your Elastic Beanstalk application won't automatically receive updated values if the secret store data is updated or rotated. Elastic Beanstalk only pulls secrets into environment variables at the time of instance bootstrapping.

Refreshing your environment variables

To trigger your Elastic Beanstalk environment to refetch the latest values of the secrets from their secret stores, we recommend that you run either the `UpdateEnvironment` or `RestartAppServer` operation. You can run these operations using the Elastic Beanstalk console, the AWS CLI, or the Elastic Beanstalk API. For more information, see *AWS CLI examples for Elastic Beanstalk*, or the AWS Elastic Beanstalk API Reference.

Managing auto scaling effects on secret synchronization

If a scale out event or instance replacement occurs after the secret store updates, the new instance that comes up will have the latest secret values from Secrets Manager or Systems Manager Parameter Store. Such an event can occur even if not all the other instances in the environment have been refreshed to retrieve the new secrets.

Important

You must ensure that your application is able to use two different secret values for the same environment variable. This accommodates events where a secret update occurs in Secrets Manager or Systems Manager Parameter Store, followed by a scale out or instance replacement in your environment, while the other instances are pending environment variable refresh. During the wait period for refresh, not all of the environment instances will have the same values for the secret store environment variables.

An example of such a use case is a database credential rotation. When a scale out event follows the credential rotation, the environment secrets referenced by the newly bootstrapped instances contain the updated database credentials. However, the environment secrets referenced by the existing instances retain the old value until they are refreshed by the `UpdateEnvironment` or `RestartAppServer` operations.

Multiline values in Amazon Linux 2 environment variables

Multiline values are composed of more than one line and include a newline character. With the exception of Docker and ECS-based Docker platforms, platforms that run on Amazon Linux 2 don't support multiline values for environment variables

Note

Elastic Beanstalk will fail the deployment of affected environments if it detects a multiline value.

The following options can serve as workarounds or solutions to the multiline issue:

- Upgrade your Amazon Linux 2 environment to Amazon Linux 2023. For more information, see Migration from Amazon Linux 2 to Amazon Linux 2023.
- Remove newline characters from your secret values. One example approach is to Base64 encode your values before storing them in the secret store. Your application would then need to decode the value back into the original format when it references it from the environment secret variable.
- Design your application code to retrieve the data directly from Secrets Manager or Systems Manager Parameter Store. For more information, see *Retrieving secrets* in Using Secrets Manager or *Retrieving parameters* Using Systems Manager Parameter Store.

Required IAM permissions for Elastic Beanstalk to access secrets and parameters

You must grant the necessary permissions to your environment's EC2 instances to fetch the secrets and parameters for AWS Secrets Manager and AWS Systems Manager Parameter Store. Permissions are provided to the EC2 instances via an EC2 instance profile role.

The following sections list the specific permissions that you need to add to an EC2 instance profile, depending on which service you use. Follow the steps provided in Update the permissions policy for a role in the *IAM User Guide* to add these permissions.

IAM permissions for the ECS managed Docker platform

The ECS managed Docker platform requires additional IAM permissions to the ones provided in this topic. For more information about all of the required permissions for your ECS managed Docker platform environment to support Elastic Beanstalk environment variables integration with secrets, see Execution Role ARN format.

Topics

Required IAM permissions for Secrets Manager

The following permissions grant access to fetch encrypted secrets from the AWS Secrets Manager store:

- secretsmanager:GetSecretValue
- kms:Decrypt

The permission to decrypt an AWS KMS key is only required if your secret uses a customer managed key instead of the default key. The addition of your custom key ARN adds the permission to decrypt the customer managed key.

Example policy with Secrets Manager and KMS key permissions

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-east-1:111122223333:secret:my-secret",
                "arn:aws:kms:us-east-1:111122223333:key/my-key"
            ]
        }
    ]
}
```

Required IAM permissions Systems Manager Parameter Store

The following permissions grant access to fetch encrypted parameters from the AWS Systems Manager Parameter Store:

- ssm:GetParameter
- kms:Decrypt

The permission to decrypt an AWS KMS key is only required for `SecureString` parameter types that uses a customer managed key instead of a default key. The addition of your custom key ARN adds the permission to decrypt the customer managed key. The regular parameter types that aren't encrypted, `String` and `StringList`, don’t need an AWS KMS key.

Example policy with Systems Manager and AWS KMS key permissions

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameter",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:ssm:us-east-1:111122223333:parameter/my-parameter",
                "arn:aws:kms:us-east-1:111122223333:key/my-key"
            ]
        }
    ]
}
```

Using AWS Secrets Manager and AWS Systems Manager Parameter Store

This topic provides a brief introduction of AWS Secrets Manager and AWS Systems Manager Parameter Store, pricing information, and references to learn more about creating and retrieving secrets, using both the console and programmatic options.

About Secrets Manager

AWS Secrets Manager helps you manage, retrieve, and rotate secrets throughout their lifecycles. Examples of secret data you can manage include database credentials, application credentials, OAuth tokens, and API keys. Secrets Manager enables you to configure an automatic rotation schedule for your secrets.

About Systems Manager Parameter Store

Parameter Store is a tool in AWS Systems Manager. It provides secure, hierarchical storage for configuration data management and secrets management. You can manage important configuration data as parameter values. Examples of data that you can manage with Parameter Store includes Amazon Machine Image (AMI) IDs, license codes, passwords, and database strings.

Pricing

Standard charges apply for using Secrets Manager and Systems Manager Parameter Store. For more information about pricing, see the following websites:

- AWS Secrets Manager pricing
- AWS Systems Manager pricing (select *Parameter Store* from the content list)

Topics

- Using Secrets Manager to create and retrieve secrets

- Using Systems Manager Parameter Store to create and retrieve parameters

Using Secrets Manager to create and retrieve secrets

You can create and retrieve Secrets Manager secrets using the AWS Secrets Manager console, the AWS CLI, or the AWS SDK. Refer to the following resources to learn more about different methods to create and retrieve Secrets Manager secrets.

Creating secrets

- Console – Create an AWS Secrets Manager secret (console) in the *AWS Secrets Manager User Guide*
- AWS CLI – AWSCLI in the *AWS Secrets Manager User Guide*
- AWS SDK – AWS SDK in the *AWS Secrets Manager User Guide*

Retrieving secrets

- Console – Get a secret value (console) in the *AWS Secrets Manager User Guide*
- AWS CLI – Get a secret value (AWS CLI) in the *AWS Secrets Manager User Guide*
- AWS SDK – Code examples for Secrets Manager using AWS SDKs in the *AWS SDK Code Examples Code Library*
- Other methods – Get secrets from AWS Secrets Manager in the *AWS Secrets Manager User Guide*

For more information about AWS Secrets Manager, see What is AWS Secrets Manager? in the *AWS Secrets Manager User Guide*.

Using Systems Manager Parameter Store to create and retrieve parameters

You can create and retrieve Parameter Store parameters using the AWS Systems Manager console, the AWS CLI, or the AWS SDK. Refer to the following resources to learn more about different methods to create and retrieve Parameter Store parameters.

Creating parameters

- Console – Create a Systems Manager parameter (console) in the *AWS Systems Manager User Guide*

- AWS CLI – Create a Systems Manager parameter (AWS CLI) in the *AWS Systems Manager User Guide*
- AWS SDK – Use PutParameter with an AWS SDK or AWS CLI in the *AWS SDK Code Examples Code Library*

Retrieving parameters

- Console – Searching for a parameter (console) in the *AWS Systems Manager User Guide*
- AWS CLI – Use GetParameter with an AWS SDK or AWS CLI in the *AWS SDK Code Examples Code Library*
- AWS SDK – Use GetParameter with an AWS SDK or AWS CLI in the *AWS SDK Code Examples Code Library*

For more information, see AWS Systems Manager Parameter Store in the *AWS Systems Manager User Guide*.

Troubleshooting secrets integration with Elastic Beanstalk environment variables

Event: *Instance deployment failed to get one or more secrets*

This message indicates that Elastic Beanstalk was not able to fetch one or more of the secrets specified during your application deployment.

- Check that the resources specified by the ARN values in your environment variable configuration exist.
- Confirm that your Elastic Beanstalk EC2 instance profile role has the required IAM permissions to access the resources.
- If this event was triggered through the `RestartAppServer` operation, once the issue is fixed, retry the `RestartAppServer` call to resolve the issue.
- If the event was triggered through an `UpdateEnvironment` call, retry the `UpdateEnvironment` operation.

For examples of these commands, see *AWS CLI examples for Elastic Beanstalk*. For more information about the API actions for these operations, see the *AWS Elastic Beanstalk API Reference*.

Event: *Instance deployment detected one or more multiline environment values, which are not supported for this platform*

Multiline variables are not supported for Amazon Linux 2 platforms, excluding Docker and ECS managed Docker platforms. For available options to proceed, see Multiline values.

Event: *CreateEnvironment fails when a secret is specified*

When `CreateEnvironment` fails and you have secrets as environment variables, you need to address the underlying issue and then use `UpdateEnvironment` to complete the environment setup. Do not use `RestartAppServer`, as it will not be sufficient to bring the environment up in this situation. For examples of these commands, see *AWS CLI examples for Elastic Beanstalk*. For more information about the API actions for these operations, see the *AWS Elastic Beanstalk API Reference*.

Using Elastic Beanstalk with Amazon VPC

This topic explains the benefits of using VPC endpoints with Elastic Beanstalk and the different types of configurations you can implement.

You can use an Amazon Virtual Private Cloud (Amazon VPC) to create a secure network for your Elastic Beanstalk application and related AWS resources. When you create your environment, you choose which VPC, subnets, and security groups are used for your application instances and load balancer. You can use any VPC configuration that you like as long as it meets the following requirements.

VPC requirements

- **Internet Access** – Instances can have access to the internet through one of the following methods:
  - **Public Subnet** – Instances have a public IP address and use an internet gateway to access the internet.
  - **Private Subnet** – Instances use a NAT device to access the internet.

> **Note**
>
> If you configure VPC endpoints in your VPC to connect to both the `elasticbeanstalk` and `elasticbeanstalk-health` services, internet access is optional, and is only

required if your application specifically needs it. Without VPC endpoints, your VPC must have access to the internet.
The default VPC that Elastic Beanstalk sets up for you provides internet access.

Elastic Beanstalk doesn't support proxy settings like `HTTPS_PROXY` for configuring a web proxy.

- **NTP** – Instances in your Elastic Beanstalk environment use Network Time Protocol (NTP) to synchronize the system clock. If instances are unable to communicate on UDP port 123, the clock may go out of sync, causing issues with Elastic Beanstalk health reporting. Ensure that your VPC security groups and network ACLs allow inbound and outbound UDP traffic on port 123 to avoid these issues.

The elastic-beanstalk-samples repository provides AWS CloudFormation templates that you can use to create a VPC for use with your Elastic Beanstalk environments.

To create resources with a AWS CloudFormation template

1. Clone the samples repository or download a template using the links in the README.
2. Open the AWS CloudFormation console.
3. Choose **Create stack**.
4. Choose **Upload a template to Amazon S3**.
5. Choose **Upload file** and upload the template file from your local machine.
6. Choose **Next** and follow the instructions to create a stack with the resources in the template.

When stack creation completes, check the **Outputs** tab to find the VPC ID and subnet IDs. Use these to configure the VPC in the new environment wizard network configuration category.

Topics

- Using endpoint policies to control access with VPC endpoints

Public VPC

AWS CloudFormation template – vpc-public.yaml

Settings (load balanced)

- **Load balancer visibility** – Public
- **Load balancer subnets –** Both public subnets
- **Instance public IP** – Enabled
- **Instance subnets –** Both public subnets
- **Instance security groups –** Add the default security group

Settings (single instance)

- **Instance subnets –** One of the public subnets
- **Instance security groups** – Add the default security group

A basic *public-only* VPC layout includes one or more public subnets, an internet gateway, and a default security group that allows traffic between resources in the VPC. When you create an environment in the VPC, Elastic Beanstalk creates additional resources that vary depending on the environment type.

VPC resources

- **Single instance** – Elastic Beanstalk creates a security group for the application instance that allows traffic on port 80 from the internet, and assigns the instance an Elastic IP to give it a public IP address. The environment's domain name resolves to the instance's public IP address.
- **Load balanced** – Elastic Beanstalk creates a security group for the load balancer that allows traffic on port 80 from the internet, and a security group for the application instances that allows traffic from the load balancer's security group. The environment's domain name resolves to the load balancer's public domain name.

This is similar to the way that Elastic Beanstalk manages networking when you use the default VPC. Security in a public subnet depends on the load balancer and instance security groups created by Elastic Beanstalk. It is the least expensive configuration as it does not require a NAT Gateway.

Public/private VPC

AWS CloudFormation template – vpc-privatepublic.yaml

Settings (load balanced)

- **Load balancer visibility** – Public
- **Load balancer subnets** – Both public subnets
- **Instance public IP** – Disabled
- **Instance subnets** – Both private subnets
- **Instance security groups** – Add the default security group

For additional security, add private subnets to your VPC to create a *public-private* layout. This layout requires a load balancer and NAT gateway in the public subnets, and lets you run your application instances, database, and any other resources in private subnets. Instances in private subnets can only communicate with the internet through the load balancer and NAT gateway.

Private VPC

AWS CloudFormation template – vpc-private.yaml

Settings (load balanced)

- **Load balancer visibility** – Private
- **Load balancer subnets** – Both private subnets
- **Instance public IP** – Disabled
- **Instance subnets** – Both private subnets
- **Instance security groups** – Add the default security group

For internal applications that shouldn't have access from the internet, you can run everything in private subnets and configure the load balancer to be internally facing (change **Load balancer**

visibility to **Internal**). This template creates a VPC with no public subnets and no internet gateway. Use this layout for applications that should only be accessible from the same VPC or an attached VPN.

Running an Elastic Beanstalk environment in a private VPC

When you create your Elastic Beanstalk environment in a private VPC, the environment doesn't have access to the internet. Your application might need access to the Elastic Beanstalk service or other services. Your environment might use enhanced health reporting, and in this case the environment instances send health information to the enhanced health service. And Elastic Beanstalk code on environment instances sends traffic to other AWS services, and other traffic to non-AWS endpoints (for example, to download dependency packages for your application). Here are some steps you might need to take in this case to ensure that your environment works properly.

- *Configure VPC endpoints for Elastic Beanstalk* – Elastic Beanstalk and its enhanced health service support VPC endpoints, which ensure that traffic to these services stays inside the Amazon network and doesn't require internet access. For more information, see the section called “VPC endpoints”.
- *Configure VPC endpoints for additional services* – Elastic Beanstalk instances send traffic to several other AWS services on your behalf: Amazon Simple Storage Service (Amazon S3), Amazon Simple Queue Service (Amazon SQS), AWS CloudFormation, and Amazon CloudWatch Logs. You must configure VPC endpoints for these services too. For detailed information about VPC endpoints, including per-service links, see VPC Endpoints in the *Amazon VPC User Guide*.

  Note

  Some AWS services, including Elastic Beanstalk, support VPC endpoints in a limited number of AWS Regions. When you design your private VPC solution, verify that Elastic Beanstalk and the other dependent services mentioned here support VPC endpoints in the AWS Region that you choose.

- *Provide a private Docker image* – In a Docker environment, code on the environment's instances might try to pull your configured Docker image from the internet during environment creation and fail. To avoid this failure, build a custom Docker image on your environment, or use a Docker image stored in Amazon Elastic Container Registry (Amazon ECR) and configure a VPC endpoint for the Amazon ECR service.
- *Enable DNS names* – Elastic Beanstalk code on environment instances sends traffic to all AWS services using their public endpoints. To ensure that this traffic goes through, choose the **Enable**

DNS name option when you configure all interface VPC endpoints. This adds a DNS entry in your VPC that maps the public service endpoint to the interface VPC endpoint.

> **Important**
>
> If your VPC isn't private and has public internet access, and if **Enable DNS name** is disabled for any VPC endpoint, traffic to the respective service travels through the public internet. This is probably not what you intend. It's easy to detect this issue with a private VPC, because it prevents this traffic from going through and you receive errors. However, with a public facing VPC, you get no indication.

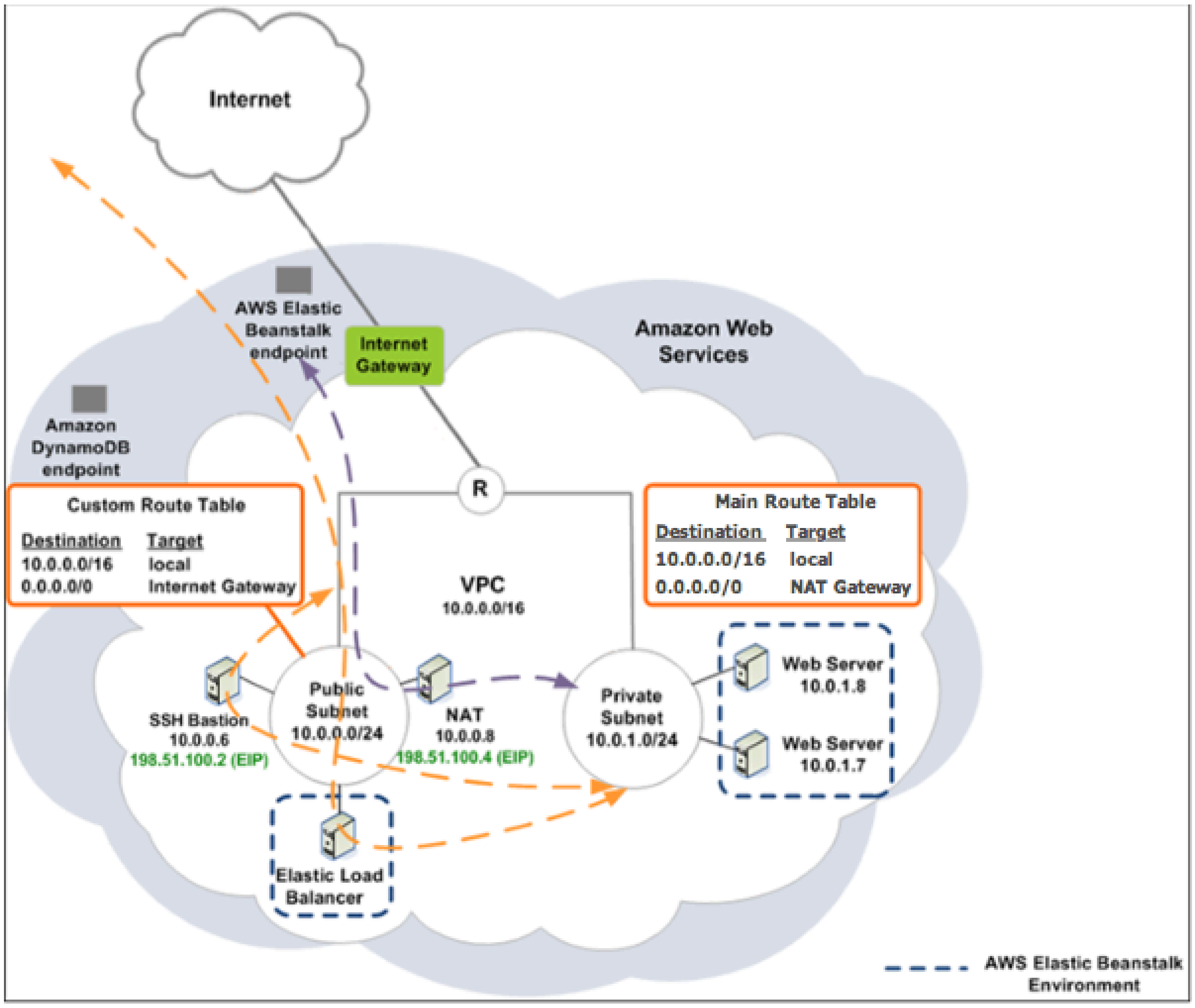
- *Include application dependencies* – If your application has dependencies such as language runtime packages, it might try to download and install them from the internet during environment creation and fail. To avoid this failure, include all dependency packages in your application's source bundle.
- *Use a current platform version* – Be sure that your environment uses a platform version that was released on February 24, 2020 or later. Specifically, use a platform version that was released in or after one of these two updates: Linux Update 2020-02-28, Windows Update 2020-02-24.

> **Note**
>
> The reason for needing an updated platform version is that older versions had an issue that would prevent DNS entries created by the **Enable DNS name** option from working properly for Amazon SQS.

Example: Launching an Elastic Beanstalk application in a VPC with bastion hosts

This section explains how to deploy an Elastic Beanstalk application inside a VPC using a bastion host and why you would implement this topology.

If your Amazon EC2 instances are located inside a private subnet, you will not be able to connect to them remotely. To connect to your instances, you can set up bastion servers in the public subnet to act as proxies. For example, you can set up SSH port forwarders or RDP gateways in the public subnet to proxy the traffic going to your database servers from your own network. This section provides an example of how to create a VPC with a private and public subnet. The instances are located inside the private subnet, and the bastion host, NAT gateway, and load balancer are located inside the public subnet. Your infrastructure will look similar to the following diagram.



To deploy an Elastic Beanstalk application inside a VPC using a bastion host, complete the steps described in the following subsections.

Steps

- Create a VPC with a public and private subnet
- Create and configure the bastion host security group
- Update the instance security group
- Create a bastion host

Create a VPC with a public and private subnet

Complete all of the procedures in Public/private VPC. When deploying the application, you must specify an Amazon EC2 key pair for the instances so you can connect to them remotely. For more information about how to specify the instance key pair, see The Amazon EC2 instances for your Elastic Beanstalk environment.

Create and configure the bastion host security group

Create a security group for the bastion host, and add rules that allow inbound SSH traffic from the Internet, and outbound SSH traffic to the private subnet that contains the Amazon EC2 instances.

To create the bastion host security group

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose **Security Groups**.
3. Choose **Create Security Group**.
4. In the **Create Security Group** dialog box, enter the following and choose **Yes, Create**.

   Name tag (Optional)

      Enter a name tag for the security group.

   Group name

      Enter the name of the security group.

   Description

      Enter a description for the security group.

   VPC

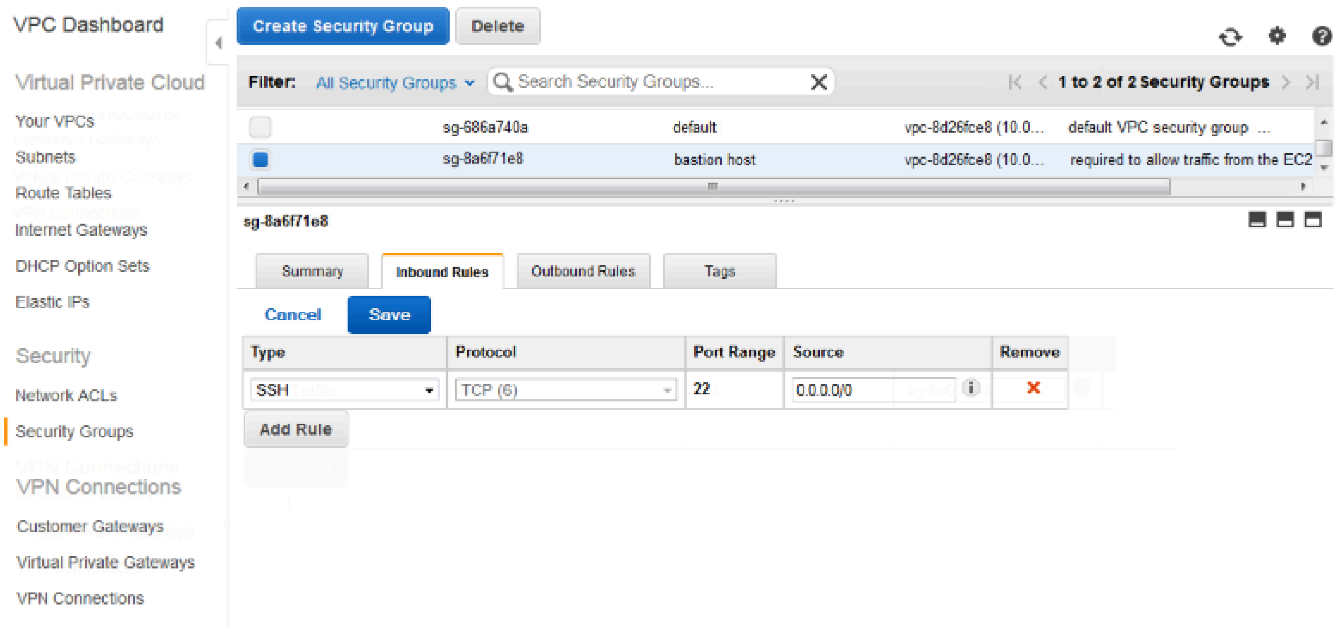
      Select your VPC.

   The security group is created and appears on the **Security Groups** page. Notice that it has an ID (e.g., `sg-xxxxxxxx`). You might have to turn on the **Group ID** column by clicking **Show/Hide** in the top right corner of the page.

To configure the bastion host security group

1. In the list of security groups, select the check box for the security group you just created for your bastion host.
2. On the **Inbound Rules** tab, choose **Edit**.
3. If needed, choose **Add another rule**.
4. If your bastion host is a Linux instance, under **Type**, select **SSH**.

   If your bastion host is a Windows instance, under **Type**, select **RDP**.
5. Enter the desired source CIDR range in the **Source** field and choose **Save**.



sg-686a740a	default	vpc-8d26fce8 (10.0...	default VPC security group ...
sg-8a6f71e8	bastion host	vpc-8d26fce8 (10.0...	required to allow traffic from the EC2

sg-8a6f71e8

Type	Protocol	Port Range	Source	Remove
SSH	TCP (6)	22	0.0.0.0/0	✕

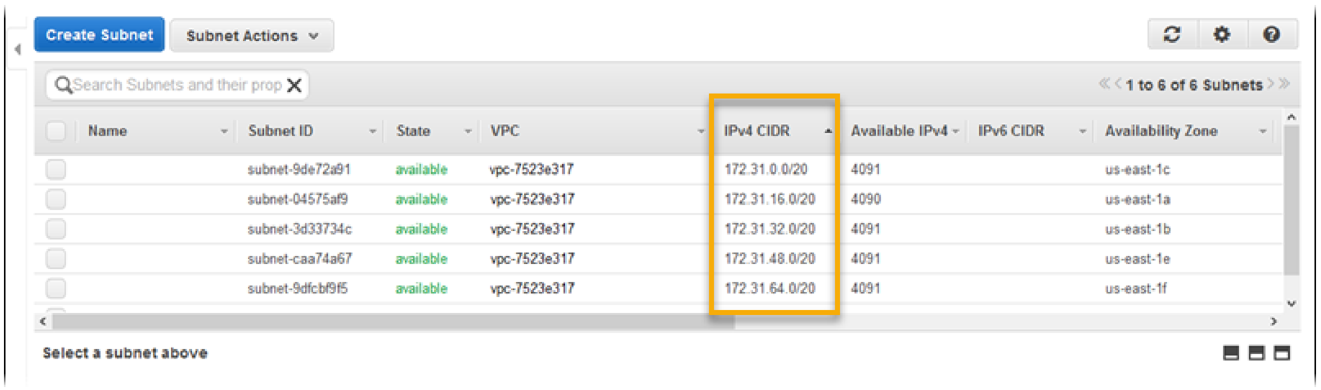
6. On the **Outbound Rules** tab, choose **Edit**.
7. If needed, choose **Add another rule**.
8. Under **Type**, select the type that you specified for the inbound rule.
9. In the **Source** field, enter the CIDR range of the subnet of the hosts in the VPC's private subnet.

   To find it:

   a. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
   b. In the navigation pane, choose **Subnets**.
   c. Note the value under **IPv4 CIDR** for each **Availability Zone** in which you have hosts that you want the bastion host to bridge to.

Note

If you have hosts in multiple availability zones, create an outbound rule for each one of these availability zones.



Name	Subnet ID	State	VPC	IPv4 CIDR	Available IPv4	IPv6 CIDR	Availability Zone
	subnet-9de72a91	available	vpc-7523e317	172.31.0.0/20	4091		us-east-1c
	subnet-04575af9	available	vpc-7523e317	172.31.16.0/20	4090		us-east-1a
	subnet-3d33734c	available	vpc-7523e317	172.31.32.0/20	4091		us-east-1b
	subnet-caa74a67	available	vpc-7523e317	172.31.48.0/20	4091		us-east-1e
	subnet-9dfcbf9f5	available	vpc-7523e317	172.31.64.0/20	4091		us-east-1f

10. Choose **Save**.

Update the instance security group

By default, the security group you created for your instances does not allow incoming traffic. While Elastic Beanstalk will modify the default group for the instances to allow SSH traffic, you must modify your custom instance security group to allow RDP traffic if your instances are Windows instances.

To update the instance security group for RDP

1. In the list of security groups, select the check box for the instance security group.
2. On the **Inbound** tab, choose **Edit**.
3. If needed, choose **Add another rule**.
4. Enter the following values, and choose **Save**.

   Type

      RDP

   Protocol

      TCP

Port Range

`3389`

Source

Enter the ID of the bastion host security group (e.g., `sg-8a6f71e8`) and choose **Save**.

Create a bastion host

To create a bastion host, you launch an Amazon EC2 instance in your public subnet that will act as the bastion host.

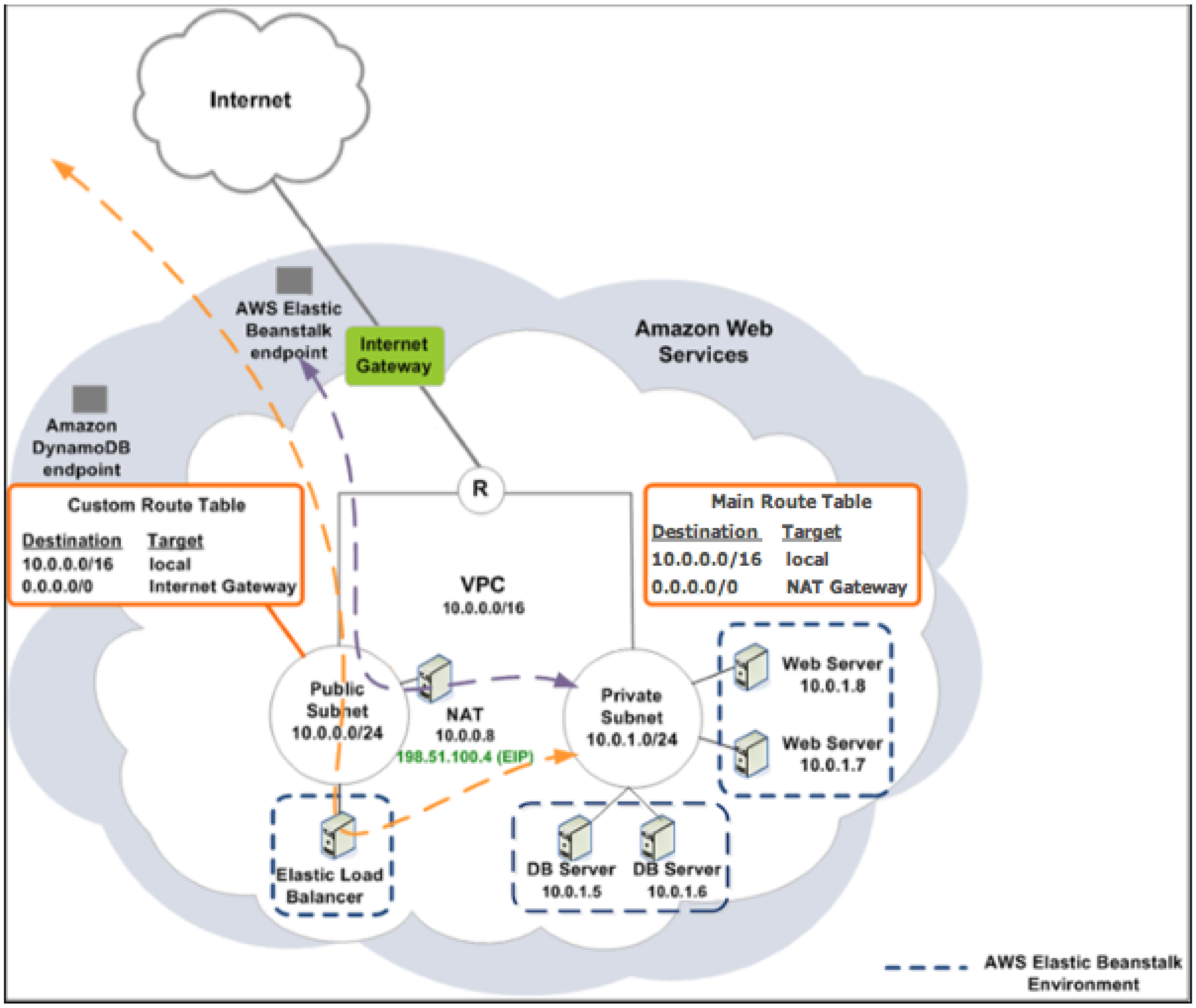
For more information about setting up a bastion host for Windows instances in the private subnet, see Controlling Network Access to EC2 Instances Using a Bastion Server.

For more information about setting up a bastion host for Linux instances in the private subnet, see Securely Connect to Linux Instances Running in a Private Amazon VPC.

Example: Launching an Elastic Beanstalk in a VPC with Amazon RDS

This section walks you through the tasks to deploy an Elastic Beanstalk application with Amazon RDS in a VPC using a NAT gateway.

Your infrastructure will look similar to the following diagram.



Note

If you haven't used a DB instance with your application before, try adding one to a test environment, and connecting to an external DB instance before adding a VPC configuration to the mix.

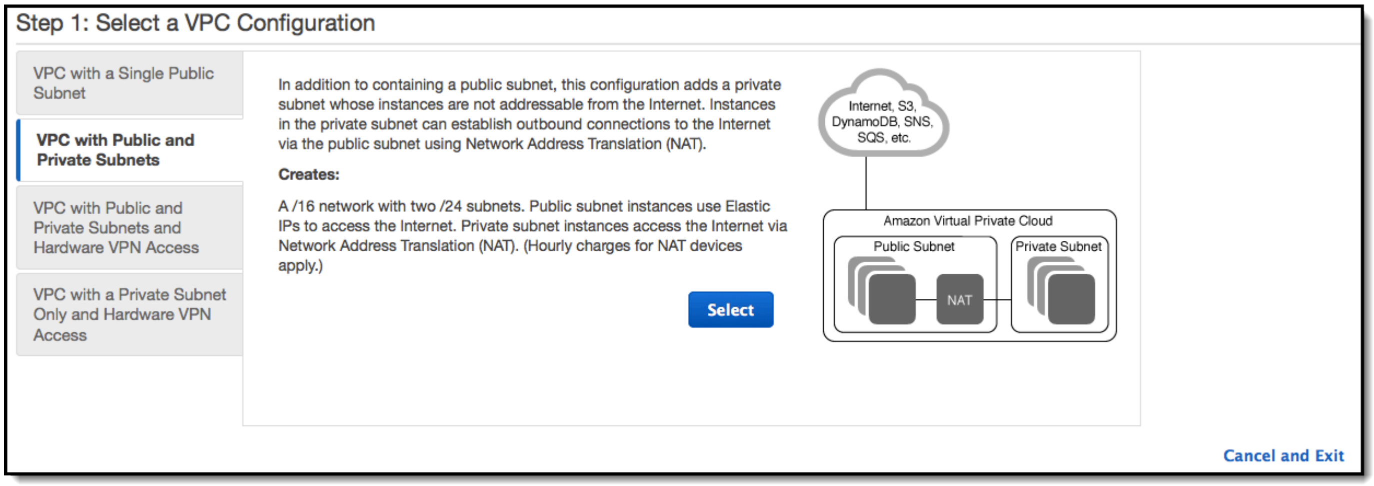
Create a VPC with a public and private subnet

You can use the Amazon VPC console to create a VPC.

To create a VPC

1. Sign in to the Amazon VPC console.
2. In the navigation pane, choose **VPC Dashboard**. Then choose **Create VPC**.

3. Choose **VPC with Public and Private Subnets**, and then choose **Select**.



Step 1: Select a VPC Configuration

VPC with a Single Public Subnet
VPC with Public and Private Subnets
VPC with Public and Private Subnets and Hardware VPN Access
VPC with a Private Subnet Only and Hardware VPN Access

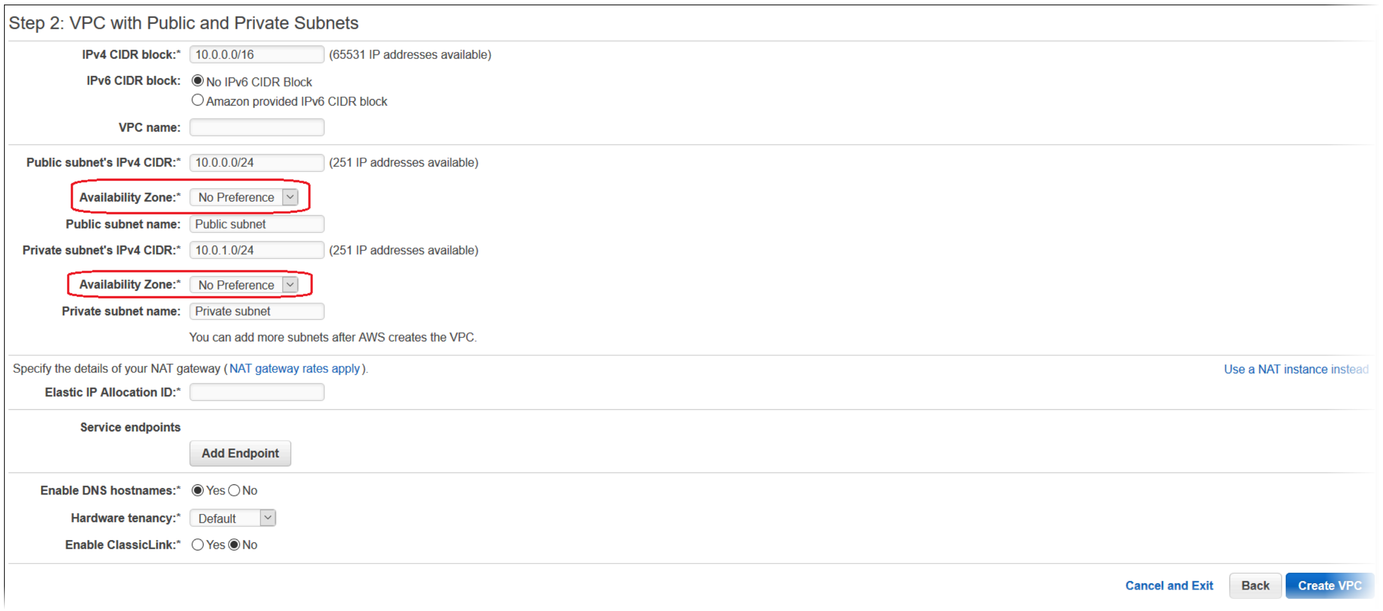
In addition to containing a public subnet, this configuration adds a private subnet whose instances are not addressable from the Internet. Instances in the private subnet can establish outbound connections to the Internet via the public subnet using Network Address Translation (NAT).

Creates:

A /16 network with two /24 subnets. Public subnet instances use Elastic IPs to access the Internet. Private subnet instances access the Internet via Network Address Translation (NAT). (Hourly charges for NAT devices apply.)

Select
Internet, S3, DynamoDB, SNS, SQS, etc.
Amazon Virtual Private Cloud
Public Subnet
Private Subnet
NAT
Cancel and Exit

4. Your Elastic Load Balancing load balancer and your Amazon EC2 instances must be in the same Availability Zone so they can communicate with each other. Choose the same Availability Zone from each **Availability Zone** list.



Step 2: VPC with Public and Private Subnets

IPv4 CIDR block:*	10.0.0.0/16 (65531 IP addresses available)
IPv6 CIDR block:	[✓ No IPv6 CIDR Block] Amazon provided IPv6 CIDR block
VPC name:	
Public subnet's IPv4 CIDR:*	10.0.0.0/24 (251 IP addresses available)
Availability Zone:*	No Preference
Public subnet name:	Public subnet
Private subnet's IPv4 CIDR:*	10.0.1.0/24 (251 IP addresses available)
Availability Zone:*	No Preference
Private subnet name:	Private subnet

You can add more subnets after AWS creates the VPC.

Specify the details of your NAT gateway (NAT gateway rates apply). Use a NAT instance instead

Elastic IP Allocation ID:*	
Service endpoints	Add Endpoint
Enable DNS hostnames:*	[✓ Yes] No
Hardware tenancy:*	Default
Enable ClassicLink:*	Yes [✓ No]

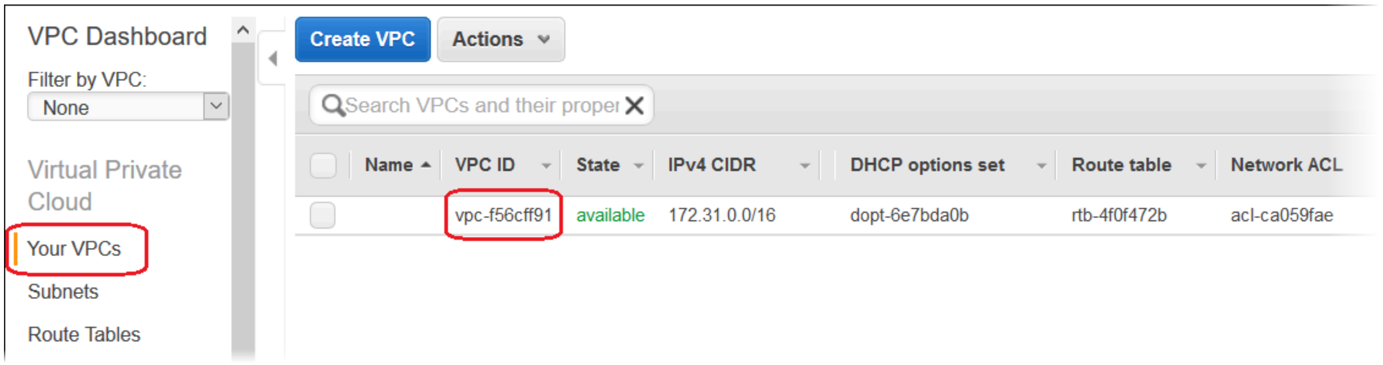
5. Choose an Elastic IP address for your NAT gateway.

6. Choose **Create VPC**.

The wizard begins to create your VPC, subnets, and internet gateway. It also updates the main route table and creates a custom route table. Finally, the wizard creates a NAT gateway in the public subnet.

> **Note**
>
> You can choose to launch a NAT instance in the public subnet instead of a NAT gateway. For more information, see Scenario 2: VPC with Public and Private Subnets (NAT) in the *Amazon VPC User Guide*.

7. After the VPC is successfully created, you get a VPC ID. You need this value for the next step. To view your VPC ID, choose **Your VPCs** in the left pane of the Amazon VPC console.



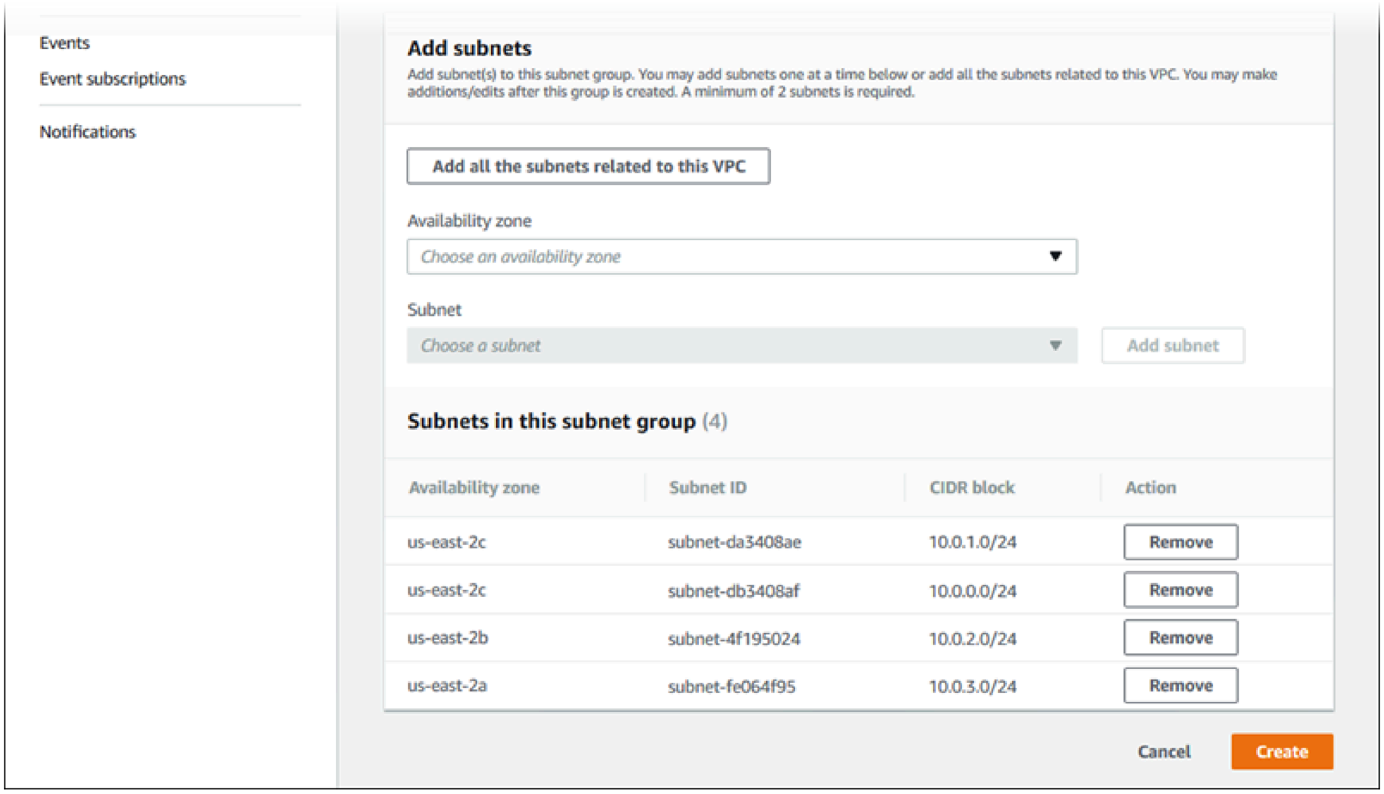
	Name	VPC ID	State	IPv4 CIDR	DHCP options set	Route table	Network ACL
		vpc-f56cff91	available	172.31.0.0/16	dopt-6e7bda0b	rtb-4f0f472b	acl-ca059fae

Create a DB subnet group

A DB subnet group for a VPC is a collection of subnets (typically private) that you can designate for your backend RDS DB instances. Each DB subnet group should have at least one subnet for every Availability Zone in a given AWS Region. To learn more, see Creating a Subnet in Your VPC.

Create a DB subnet group

1. Open the Amazon RDS console.
2. In the navigation pane, choose **Subnet groups**.
3. Choose **Create DB Subnet Group**.
4. Choose **Name**, and then type the name of your DB subnet group.
5. Choose **Description**, and then describe your DB subnet group.
6. For **VPC**, choose the ID of the VPC that you created.
7. In **Add subnets**, choose **Add all the subnets related to this VPC**.



Add subnets

Add subnet(s) to this subnet group. You may add subnets one at a time below or add all the subnets related to this VPC. You may make additions/edits after this group is created. A minimum of 2 subnets is required.

Add all the subnets related to this VPC
Availability zone
Choose an availability zone
Subnet
Choose a subnet
Add subnet

Subnets in this subnet group (4)

Availability zone	Subnet ID	CIDR block	Action
us-east-2c	subnet-da3408ae	10.0.1.0/24	Remove
us-east-2c	subnet-db3408af	10.0.0.0/24	Remove
us-east-2b	subnet-4f195024	10.0.2.0/24	Remove
us-east-2a	subnet-fe064f95	10.0.3.0/24	Remove

8. When you are finished, choose **Create**.

   Your new DB subnet group appears in the Subnet groups list of the Amazon RDS console. You can choose it to see details, such as all of the subnets associated with this group, in the details pane at the bottom of the page.

Deploy to Elastic Beanstalk

After you set up your VPC, you can create your environment inside it and deploy your application to Elastic Beanstalk. You can do this using the Elastic Beanstalk console, or you can use the AWS toolkits, AWS CLI, EB CLI, or Elastic Beanstalk API. If you use the Elastic Beanstalk console, you just need to upload your `.war` or `.zip` file and select the VPC settings inside the wizard. Elastic Beanstalk then creates your environment inside your VPC and deploys your application. Alternatively, you can use the AWS toolkits, AWS CLI, EB CLI, or Elastic Beanstalk API to deploy your application. To do this, you need to define your VPC option settings in a configuration file and deploy this file with your source bundle. This topic provides instructions for both methods.

Deploying with the Elastic Beanstalk console

The Elastic Beanstalk console walks you through creating your new environment inside your VPC. You need to provide a `.war` file (for Java applications) or a `.zip` file (for all other applications).

On the **VPC Configuration** page of the Elastic Beanstalk environment wizard, you must make the following selections:

VPC

Select your VPC.

VPC security group

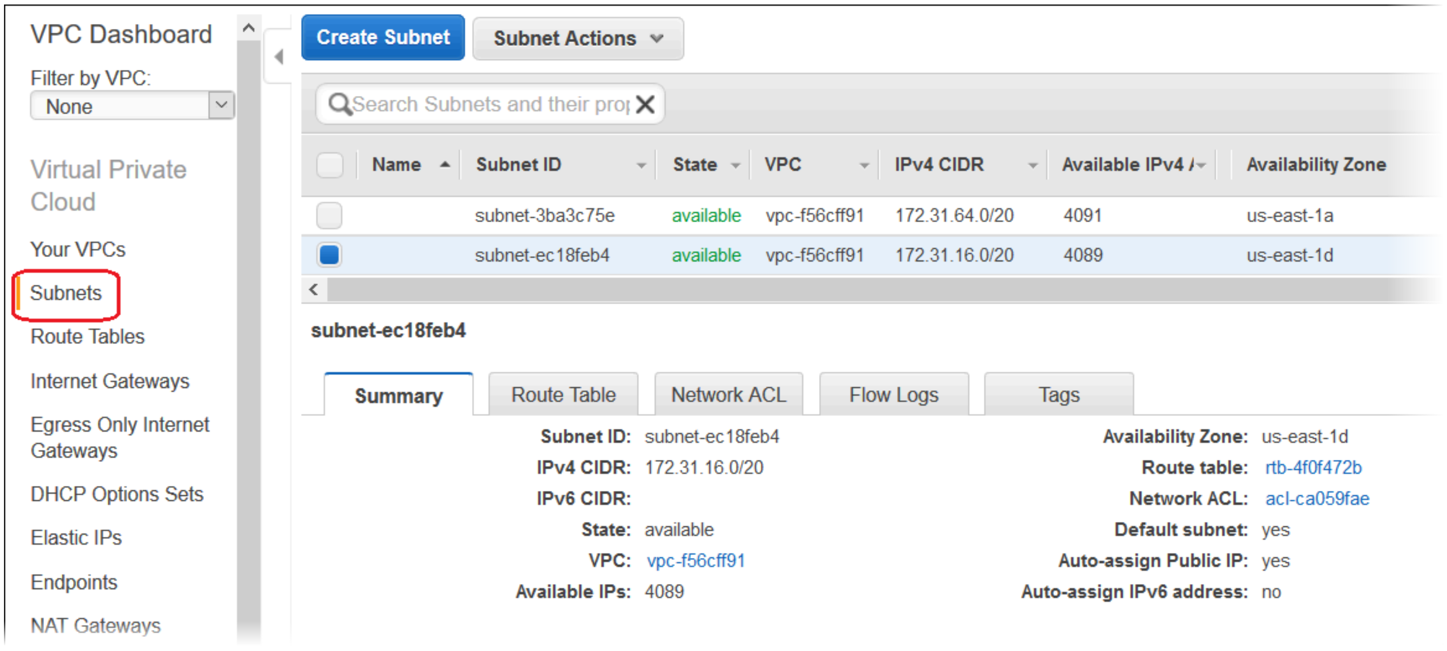
Select the instance security group you created above.

ELB visibility

Select `External` if your load balancer should be publicly available, or select `Internal` if the load balancer should be available only within your VPC.

Select the subnets for your load balancer and EC2 instances. Be sure you select the public subnet for the load balancer, and the private subnet for your Amazon EC2 instances. By default, the VPC creation wizard creates the public subnet in `10.0.0.0/24` and the private subnet in `10.0.1.0/24`.

You can view your subnet IDs by choosing **Subnets** in the [Amazon VPC console](https://console.aws.amazon.com/vpc/).



	Name	Subnet ID	State	VPC	IPv4 CIDR	Available IPv4 A	Availability Zone
☐		subnet-3ba3c75e	available	vpc-f56cff91	172.31.64.0/20	4091	us-east-1a
■		subnet-ec18feb4	available	vpc-f56cff91	172.31.16.0/20	4089	us-east-1d

subnet-ec18feb4

Subnet ID:	subnet-ec18feb4	**Availability Zone:**	us-east-1d
IPv4 CIDR:	172.31.16.0/20	**Route table:**	rtb-4f0f472b
IPv6 CIDR:		**Network ACL:**	acl-ca059fae
State:	available	**Default subnet:**	yes
VPC:	vpc-f56cff91	**Auto-assign Public IP:**	yes
Available IPs:	4089	**Auto-assign IPv6 address:**	no

Deploying with the AWS toolkits, EB CLI, AWS CLI, or API

When deploying your application to Elastic Beanstalk using the AWS toolkits, EB CLI, AWS CLI, or API, you can specify your VPC option settings in a file and deploy it with your source bundle.

See Advanced environment customization with configuration files (`.ebextensions`) for more information.

When you update the option settings, you need to specify at least the following:

- **VPCId**–Contains the ID of the VPC.
- **Subnets**–Contains the ID of the Auto Scaling group subnet. In this example, this is the ID of the private subnet.
- **ELBSubnets**–Contains the ID of the subnet for the load balancer. In this example, this is the ID of the public subnet.
- **SecurityGroups**–Contains the ID of the security groups.
- **DBSubnets**–Contains the ID of the DB subnets.

> **Note**
>
> When using DB subnets, you need to create additional subnets in your VPC to cover all the Availability Zones in the AWS Region.

Optionally, you can also specify the following information:

- **ELBScheme** – Specify `internal` to create an internal load balancer inside your VPC so that your Elastic Beanstalk application can't be accessed from outside your VPC.

The following is an example of the option settings you could use when deploying your Elastic Beanstalk application inside a VPC. For more information about VPC option settings (including examples for how to specify them, default values, and valid values), see the **aws:ec2:vpc** namespace table in Configuration options.

```
option_settings:
  - namespace: aws:autoscaling:launchconfiguration
    option_name: EC2KeyName
    value: ec2keypair

  - namespace: aws:ec2:vpc
    option_name: VPCId
    value: vpc-170647c

  - namespace: aws:ec2:vpc
```

```
    option_name: Subnets
    value: subnet-4f195024

  - namespace: aws:ec2:vpc
    option_name: ELBSubnets
    value: subnet-fe064f95

  - namespace: aws:ec2:vpc
    option_name: DBSubnets
    value: subnet-fg148g78

  - namespace: aws:autoscaling:launchconfiguration
    option_name: InstanceType
    value: m1.small

  - namespace: aws:autoscaling:launchconfiguration
    option_name: SecurityGroups
    value: sg-7f1ef110
```

Note

When using DB subnets, be sure you have subnets in your VPC to cover all the Availability Zones in the AWS Region.

Using Elastic Beanstalk with VPC endpoints

This topic explains the benefits that a VPC endpoint can offer your Elastic Beanstalk application. It also provides instructions to create an interface VPC endpoint to an Elastic Beanstalk service.

A VPC endpoint enables you to privately connect your VPC to supported AWS services and VPC endpoint services powered by AWS PrivateLink, without requiring an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection.

Instances in your VPC don't require public IP addresses to communicate with resources in the service. Traffic between your VPC and the other service doesn't leave the Amazon network. For complete information about VPC endpoints, see VPC Endpoints in the *Amazon VPC User Guide*.

AWS Elastic Beanstalk supports AWS PrivateLink, which provides private connectivity to the Elastic Beanstalk service and eliminates exposure of traffic to the public internet. To enable your application to send requests to Elastic Beanstalk using AWS PrivateLink, you configure a type of

VPC endpoint known as an *interface VPC endpoint* (interface endpoint). For more information, see Interface VPC Endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

Note

Elastic Beanstalk supports AWS PrivateLink and interface VPC endpoints in a limited number of AWS Regions. We're working to extend support to more AWS Regions in the near future.

IPv6 support

Elastic Beanstalk supports incoming traffic over IPv4 and IPv6. This section describes the public endpoints that support IPV6 and also explains how to configure your Elastic Beanstalk VPC endpoints to support dual-stack traffic.

For more general information about IPv6, see AWS services that support IPv6 in the *Amazon VPC User Guide* and the AWS whitepaper IPv6 on AWS.

Public endpoints

The Elastic Beanstalk service has two sets of endpoints that consists of the older IPv4 endpoints and the more recent endpoints with dual-stack capability. Both sets of endpoints follow AWS naming standards:

- **IPv4** endpoints use the domain `amazonaws.com` – format for general service endpoint: `elasticbeanstalk.`*`region`*`.amazonaws.com`
- **Dual-stack** endpoints use the domain `api.aws` – format for general service endpoint:: `elasticbeanstalk.`*`region`*`.api.aws`

The endpoints for *service health* and *FIPS* have different host names, but they follow the same domain name pattern. For a list of endpoints see Elastic Beanstalk service endpoints in the *Amazon Web Services General Reference*.

Requests to Elastic Beanstalk

When you send requests to the Elastic Beanstalk service with the AWS CLI or the AWS SDK you can specify an IPv4 endpoint or a dual-stack endpoint. The AWS CLI and AWS SDK use the IPv4-only endpoints by default if an endpoint URL isn't specified.

The following example demonstrates the AWS CLI sending a request to a dual-stack endpoint:

Example

```
aws elasticbeanstalk list-available-solution-stacks \
    --endpoint-url "https://elasticbeanstalk.us-east-1.api.aws"
```

The following example demonstrates the AWS Python SDK sending a request to a dual-stack endpoint:

Example

```
import boto3

dual_stack_eb_client = boto3.client(
    service_name='elasticbeanstalk',
    region_name='us-east-1',
    endpoint_url='https://elasticbeanstalk.us-east-1.api.aws';
)

print(dual_stack_eb_client.list_available_solution_stacks())
```

VPC endpoints for dual-stack IPs

To configure your Elastic Beanstalk VPC endpoints to support dual-stack traffic, specify **dualstack** for the **IP address type** parameter of the VPC endpoint. You can specify this field via the AWS CLI, the AWS SDK, or the AWS PrivateLink console. For instructions to do so in the AWS PrivateLink console, see Create a VPC endpoint in the *AWS PrivateLink Guide*.

Note

You must specify the **IP address type** of the VPC endpoint as either **IPv4** or **dualstack**. At this time Elastic Beanstalk VPC endpoints don't support an **IP address type** of **IPv6**, which would indicate IPv6-only support. The **dualstack** option allows for both the IPv4 and IPv6 internet protocols.

The following example demonstrates how to create a dual-stack VPC endpoint with the AWS CLI:

Example

```
aws ec2 create-vpc-endpoint \
    --vpc-id "vpc-example"
    --service-name "com.amazonaws.us-east-1.elasticbeanstalk"
    --ip-address-type "dualstack"
```

Setting up a VPC endpoint for Elastic Beanstalk

To create the interface VPC endpoint for the Elastic Beanstalk service in your VPC, follow the Creating an Interface Endpoint procedure.

- For **Service Name**, choose **com.amazonaws.*region*.elasticbeanstalk**.
- For **IP address type**, choose either **IPv4** or **Dualstack**. At this time Elastic Beanstalk VPC endpoints don't support an **IP address type** of **IPv6**, which would indicate IPv6-only support. The **Dualstack** option allows for both the IPv4 and IPv6 internet protocols.

If your VPC is configured with public internet access, your application can still access Elastic Beanstalk over the internet using either the `elasticbeanstalk.region.amazonaws.com` or the `elasticbeanstalk.region.api.aws` public endpoint. You can prevent this by ensuring that **Enable DNS name** is enabled during endpoint creation (true by default). This adds a DNS entry in your VPC that maps the public service endpoint to the interface VPC endpoint.

Setting up a VPC endpoint for enhanced health

If you enabled enhanced health reporting for your environment, you can configure enhanced health information to be sent over AWS PrivateLink too. Enhanced health information is sent by the `healthd` daemon, an Elastic Beanstalk component on your environment instances, to a separate Elastic Beanstalk enhanced health service. To create an interface VPC endpoint for this service in your VPC, follow the Creating an Interface Endpoint procedure.

- For **Service Name**, choose **com.amazonaws.*region*.elasticbeanstalk-health**.
- For **IP address type**, choose either **IPv4** or **Dualstack**. At this time Elastic Beanstalk VPC endpoints don't support an **IP address type** of **IPv6**, which would indicate IPv6-only support. The **Dualstack** option allows for both the IPv4 and IPv6 internet protocols.

Important

The `healthd` daemon sends enhanced health information to the public endpoint `elasticbeanstalk-health.`*`region`*`.amazonaws.com` or `elasticbeanstalk-health.`*`region`*`.api.aws`. If your VPC is configured with public internet access, and **Enable DNS name** is disabled for the VPC endpoint, enhanced health information travels through the public internet. This is probably not your intention when you set up an enhanced health VPC endpoint. Ensure that **Enable DNS name** is enabled (true by default).

Using VPC endpoints in a private VPC

A private VPC, or a private subnet in a VPC, has no public internet access. You might want to run your Elastic Beanstalk environment in a private VPC and configure interface VPC endpoints for enhanced security. In this case, be aware that your environment might try to connect to the internet for other reasons in addition to contacting the Elastic Beanstalk service. To learn more about running an environment in a private VPC, see the section called "Running an Elastic Beanstalk environment in a private VPC".

Using endpoint policies to control access with VPC endpoints

This topic explains how you can attach a policy to VPC endpoints to controls access to your application (your service) and your Elastic Beanstalk environment.

An endpoint policy is an AWS Identity and Access Management (IAM) resource policy that controls access from the endpoint to the specified service. The endpoint policy is specific to the endpoint. It's separate from any user or instance IAM policies that your environment might have and doesn't override or replace them.

By default, a VPC endpoint allows full access to the service with which it's associated. When you create or modify an endpoint, you can attach an *endpoint policy* to it to control access to specific resources associated with the service. For details about authoring and using VPC endpoint policies, see Control access to VPC endpoints using endpoint policies in the *AWS PrivateLink Guide*.

Note

When you create restrictive endpoint policies you may need to add specific permissions to required resources, so that access to these resources isn't blocked by the endpoint policy. Doing so ensures that your environment continues to deploy and function properly.

The following example denies all users the permission to terminate an environment through the VPC endpoint, and allows full access to all other actions.

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        },
        {
            "Action": "elasticbeanstalk:TerminateEnvironment",
            "Effect": "Deny",
            "Resource": "*",
            "Principal": "*"
        }
    ]
}
```

Required Amazon S3 bucket permissions for restrictive VPC endpoint policies

If you add restrictions to your VPC endpoint policies, you must include specific Amazon S3 bucket permissions to ensure that your environment continues to deploy and function properly. This section explains the required S3 buckets and includes example policies.

Topics

S3 Buckets that store assets to manage environment platforms

The Elastic Beanstalk service owns S3 buckets that store the assets associated with a solution stack (platform version). These assets include configuration files, the sample application, and available instance types. When Elastic Beanstalk creates and manages your environment it retrieves the required information for the specific platform version from the asset bucket for each corresponding AWS Region.

S3 Bucket ARN

`arn:aws:s3:::elasticbeanstalk-samples-`*`region`*

Amazon Linux 2 and later

- `arn:aws:s3:::elasticbeanstalk-platform-assets-`*`region`*

> **Note**
>
> The bucket name follows a different convention for the *BJS* region. The string *public-beta-cn-north-1* is used in place of *`region`*. For example, `arn:aws:s3:::elasticbeanstalk-platform-assets-public-beta-cn-north-1`.

Windows Server, Amazon Linux (AMI), Amazon Linux 2 and later

- `arn:aws:s3:::elasticbeanstalk-env-resources-`*`region`*
- `arn:aws:s3:::elasticbeanstalk-`*`region`*

> **Note**
>
> The bucket names for platform-assets and env-resources buckets follow different conventions in some regions. See the region-specific bucket naming patterns section below for details.

Region-specific bucket ARN patterns

Region	Bucket Type	Bucket ARN Pattern
me-central-1	platform-assets	`arn:aws:s3:::elasticbeanstalk-platform-assets-me-central-1-f08b818c`
	env-resources	`arn:aws:s3:::elasticbeanstalk-env-resources-me-central-1-f08b818c`

Operations

GetObject

VPC endpoint policy example

The following example illustrates how to provide access to the S3 buckets required for Elastic Beanstalk operations in the US East (Ohio) Region (us-east-2). The example lists all of the buckets for both Amazon Linux and Windows Server platforms. Update your policy to only include the buckets that apply to the operating system of your environment.

Example policy

Important

We recommend that you avoid using wildcard characters (*) in place of specific Regions in this policy. For example, use `arn:aws:s3:::cloudformation-waitcondition-us-east-2/*` and don't use `arn:aws:s3:::cloudformation-waitcondition-*/*`. Using wildcards could provide access to S3 buckets that you don’t intend to grant access to. If you want to use the policy for more than one Region, we recommend repeating the first `Statement` block for each Region.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRequestsToAWSResources",
            "Effect": "Allow",
            "Principal": {"AWS": "*"},
            "Action": ["s3:GetObject"],
            "Resource": [
                "arn:aws:s3:::elasticbeanstalk-platform-assets-us-east-2/*",
                "arn:aws:s3:::elasticbeanstalk-env-resources-us-east-2/*",
                "arn:aws:s3:::elasticbeanstalk-env-resources-us-east-2/*",
                "arn:aws:s3:::elasticbeanstalk-samples-us-east-2/*"
            ]
        }
    ]
}
```

S3 Buckets owned by AWS CloudFormation

Elastic Beanstalk uses AWS CloudFormation to create resources for your environment. CloudFormation owns S3 buckets in each AWS Region to monitor responses to wait conditions.

Services like Elastic Beanstalk communicate with CloudFormation by sending requests to a presigned Amazon S3 URL for the S3 bucket that CloudFormation owns. CloudFormation creates the presigned Amazon S3 URL using the `cloudformation.amazonaws.com` service principal.

For more detailed information, see Considerations for CloudFormation VPC endpoints in the *AWS CloudFormation User Guide*. To learn more about presigned URLs, see Working with presigned URLs in the *Amazon S3 User Guide*.

S3 Bucket ARN

- `arn:aws:s3:::cloudformation-waitcondition-`*`region`*

  When using wait conditions, region names do contain dashes. For example, *us-west-2*.

- `arn:aws:s3:::cloudformation-custom-resource-response-`*`region`*

  When using custom resources, region names don't contain dashes. For example, *uswest2*.

Operations

GetObject

VPC endpoint policy example

The following example illustrates how to provide access to the S3 buckets required for Elastic Beanstalk operations in the US East (Ohio) Region (us-east-2).

Example policy

Important

We recommend that you avoid using wildcard characters (*) in place of specific Regions in this policy. For example, use `arn:aws:s3:::cloudformation-waitcondition-us-east-2/*` and don't use `arn:aws:s3:::cloudformation-waitcondition-*/*`. Using wildcards could provide access to S3 buckets that you don’t intend to grant access to. If you want to use the policy for more than one Region, we recommend repeating the first `Statement` block for each Region.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRequestsToCloudFormation",
            "Effect": "Allow",
            "Principal": {"AWS": "*"},
            "Action": ["s3:GetObject"],
            "Resource": [
                "arn:aws:s3:::cloudformation-waitcondition-us-east-2/*",
                "arn:aws:s3:::cloudformation-custom-resource-response-us-east-2/
*"
            ]
        }
    ]
}
```

S3 Buckets owned by customer accounts to store source code and other items

This bucket is owned by the AWS customer account that owns the environment. It stores resources that are specific to your environment, such as source code and requested logs.

S3 Bucket ARN

```
arn:aws:s3:::elasticbeanstalk-region-account-id
```

Operations

- GetObject
- GetObjectAcl
- PutObject
- PutObjectAcl
- ListBucket

VPC endpoint policy example

The following example illustrates how to provide access to the S3 buckets required for Elastic Beanstalk operations in the US East (Ohio) Region (us-east-2) and for the example AWS account id 123456789012.

Example policy

Important

We recommend that you avoid using wildcard characters (*) in place of specific Regions in this policy. For example, use `arn:aws:s3:::cloudformation-waitcondition-us-east-2/*` and don't use `arn:aws:s3:::cloudformation-waitcondition-*/*`. Using wildcards could provide access to S3 buckets that you don’t intend to grant access to. If you want to use the policy for more than one Region, we recommend repeating the first `Statement` block for each Region.

S3 Buckets owned by customer accounts to support Docker registry authentication

This bucket only applies to environments based on the Docker platform. The bucket stores a file used to authenticate to a private Docker registry that resides on an S3 bucket provisioned by the customer. For more information, see Using the `Dockerrun.aws.json` v3 file in the Docker platform chapter of this guide.

S3 Bucket ARN

The ARN varies by customer account.

The S3 bucket ARN has the following format: `arn:aws:s3:::`*`bucket-name`*

Operations

GetObject

VPC endpoint policy example

The following example illustrates how to provide access to an S3 bucket with the name amzn-s3-demo-bucket1.

Example policy

Updating your VPC endpoint policy

Because a VPC endpoint has only one policy attached, you must combine all of the permissions into the one policy. The following example provides all of the previous examples combined into one.

For details about authoring and using VPC endpoint policies, see Control access to VPC endpoints using endpoint policies in the *AWS PrivateLink Guide*.

Like the previous examples, the following one illustrates how to provide access to the S3 buckets required for Elastic Beanstalk operations in the US East (Ohio) Region (us-east-2). It also includes buckets with example AWS account id 123456789012 and example bucket name amzn-s3-demo-bucket1.

Important

We recommend that you avoid using wildcard characters (*) in place of specific Regions in this policy. For example, use `arn:aws:s3:::cloudformation-waitcondition-us-east-2/*` and don't use `arn:aws:s3:::cloudformation-waitcondition-*/*`. Using wildcards could provide access to S3 buckets that you don’t intend to grant access to. If you want to use the policy for more than one Region, we recommend repeating the first `Statement` block for each Region.

AWS Elastic Beanstalk security

Use this chapter to learn more about the security tasks Elastic Beanstalk is responsible for, along with the security configurations you should consider when using Elastic Beanstalk to meet your security and compliance objectives.

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The Shared Responsibility Model describes this as *Security of the Cloud* and *Security in the Cloud*.

Security of the Cloud – AWS is responsible for protecting the infrastructure that runs all of the services offered in the AWS Cloud and providing you with services that you can use securely. Our security responsibility is the highest priority at AWS, and the effectiveness of our security is regularly tested and verified by third-party auditors as part of the AWS Compliance Programs. Review the AWS Services in Scope of AWS assurance programs for information as it relates to Elastic Beanstalk.

Security in the Cloud – Your responsibility is determined by the AWS service you are using, and other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations. This documentation is intended to help you understand how to apply the Shared Responsibility Model when using Elastic Beanstalk.

Topics

Data protection in Elastic Beanstalk

The AWS shared responsibility model applies to data protection in AWS Elastic Beanstalk. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Elastic Beanstalk or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

For other Elastic Beanstalk security topics, see AWS Elastic Beanstalk security.

Topics

- Protecting data using encryption
- Internetwork traffic privacy

Protecting data using encryption

You can use different forms of data encryption to protect your Elastic Beanstalk data. Data protection refers to protecting data while *in transit* (as it travels to and from Elastic Beanstalk) and *at rest* (while it is stored in AWS data centers).

Encryption in transit

You can achieve data protection in transit in two ways: encrypt the connection using Secure Sockets Layer (SSL), or use client-side encryption (where the object is encrypted before it is sent). Both methods are valid for protecting your application data. To secure the connection, encrypt it using SSL whenever your application, its developers and administrators, and its end users send or receive any objects. For details about encrypting web traffic to and from your application, see the section called “HTTPS”.

Client-side encryption isn't a valid method for protecting your source code in application versions and source bundles that you upload. Elastic Beanstalk needs access to these objects, so they can't be encrypted. Therefore, be sure to secure the connection between your development or deployment environment and Elastic Beanstalk.

Encryption at rest

To protect your application's data at rest, learn about data protection in the storage service that your application uses. For example, see Data Protection in Amazon RDS in the *Amazon RDS User Guide*, Data Protection in Amazon S3 in the *Amazon Simple Storage Service User Guide*, or Encrypting Data and Metadata in EFS in the *Amazon Elastic File System User Guide*.

Elastic Beanstalk stores various objects in an encrypted Amazon Simple Storage Service (Amazon S3) bucket that it creates for each AWS Region in which you create environments. Because Elastic Beanstalk retains the default encryption provided by Amazon S3, it creates encrypted Amazon S3 buckets. For details, see the section called “Amazon S3”. You provide some of the stored objects and send them to Elastic Beanstalk, for example, application versions and source bundles. Elastic Beanstalk generates other objects, for example, log files. In addition to the data that Elastic Beanstalk stores, your application can transfer and/or store data as part of its operation.

To protect data stored on Amazon Elastic Block Store(Amazon EBS) volumes attached to your environment's instances, enable Amazon EBS encryption by default in your AWS account and Region. When enabled, all new Amazon EBS volumes and their snapshots are automatically encrypted using AWS Key Management Service keys. For more information, see Encryption by default in the *Amazon EBS User Guide*.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For other Elastic Beanstalk security topics, see AWS Elastic Beanstalk security.

Internetwork traffic privacy

You can use Amazon Virtual Private Cloud (Amazon VPC) to create boundaries between resources in your Elastic Beanstalk application and control traffic between them, your on-premises network, and the internet. For details, see the section called “Amazon VPC”.

For more information about Amazon VPC security, see Security in the *Amazon VPC User Guide*.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For other Elastic Beanstalk security topics, see AWS Elastic Beanstalk security.

Identity and access management for Elastic Beanstalk

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS Elastic Beanstalk resources. IAM is an AWS service that you can use with no additional charge.

For details on working with IAM, see Using Elastic Beanstalk with AWS Identity and Access Management.

For other Elastic Beanstalk security topics, see AWS Elastic Beanstalk security.

AWS managed policies for AWS Elastic Beanstalk

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

Elastic Beanstalk updates to AWS managed policies

View details about updates to AWS managed policies for Elastic Beanstalk since March 1, 2021.

To see the JSON source for a specific managed policy, see the *AWS Managed Policy Reference Guide*.

Change	Description	Date
AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy –Updated existing policy	This policy was updated to allow Elastic Beanstalk to perform managed updates when Tag propagation to launch templates is enabled. For more information, see Managed service role policies.	February 27, 2025
AdministratorAccess-AWSElasticBeanstalk – Updated existing policy	This policy was updated to replace the *StringLike* operator with the *ArnLike* operator to evaluate the ARN-	December 11, 2024

Change	Description	Date
	type keys in the condition block `iam:PolicyArn` . This provides more secure enforcement. For more information, see [Managing Elastic Beanstalk user policies]().	
The following polices were updated: • **AWSElasticBeanstalkInternalMaintenanceRolePolicy** • **AWSElasticBeanstalkMaintenance** • **AWSElasticBeanstalkManagedUpdatesInternalServiceRolePolicy** • **AWSElasticBeanstalkManagedUpdatesServiceRolePolicy** • **AWSElasticBeanstalkRoleCore**	These policies were updated to allow Elastic Beanstalk to add or remove tags when it creates or updates an AWS CloudFormation stack or change set. For more information about `AWSElasticBeanstalkManagedUpdatesServiceRolePolicy` , see [Service-linked role permissions for Elastic Beanstalk](). For more information about `AWSElasticBeanstalkRoleCore` , see [Policies for integration with other services]().	April 30, 2024

Change	Description	Date
AWSElasticBeanstalkService –Updated existing policy	This policy was updated to allow Elastic Beanstalk to tag resources upon creation for Elastic Load Balancing, Auto Scaling groups (ASG), and Amazon ECS. **Note** This policy has been previously superseded by `AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy` . Although this policy is no longer available for attachment to new IAM users, groups, or roles, it may still be attached to prior existing ones. For more information, see Managed service role policies.	May 10, 2023

Change	Description	Date
AWSElasticBeanstalkMulticontainerDocker – Updated existing policy	This policy was updated to allow Elastic Beanstalk to tag resources upon creation for Amazon ECS. For more information, see Managing Elastic Beanstalk instance profiles.	March 23, 2023
AWSElasticBeanstalkRoleECS –Updated existing policy	This policy was updated to allow Elastic Beanstalk to tag resources upon creation for Amazon ECS. For more information, see Policies for integration with other services.	March 23, 2023
AdministratorAccess-AWSElasticBeanstalk – Updated existing policy	This policy was updated to allow Elastic Beanstalk to tag resources upon creation for Amazon ECS. For more information, see Managing Elastic Beanstalk user policies.	March 23, 2023
AWSElasticBeanstalkManagedUpdatesServiceRolePolicy –Updated existing policy	This policy was updated to allow Elastic Beanstalk to add tags to Amazon ECS resources when it creates them. For more information, see Service-linked role permissions for Elastic Beanstalk.	March 23, 2023

Change	Description	Date
AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy –Updated existing policy	This policy was updated to allow Elastic Beanstalk to add tags to Amazon ECS resources when it creates them. For more information, see Managed service role policies.	March 23, 2023
AWSElasticBeanstalkManagedUpdatesServiceRolePolicy –Updated existing policy	This policy was updated to allow Elastic Beanstalk to add tags to Auto Scaling groups when it creates them. For more information, see The managed-updates service-linked role.	January 27, 2023
AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy –Updated existing policy	This policy was updated to allow Elastic Beanstalk to add tags on create of an Auto Scaling group (ASG). For more information, see Managed service role policies.	January 23, 2023
AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy –Updated existing policy	This policy was updated to allow Elastic Beanstalk to add tags on create of an elastic load balancer (ELB). For more information, see Managed service role policies.	December 21, 2022

Change	Description	Date
AWSElasticBeanstalkManagedUpdatesServiceRolePolicy –Updated existing policy	Permissions were added to this policy to allow Elastic Beanstalk to do the following during managed updates: • Create and delete launch templates and template versions. • Launch Amazon EC2 instances with launch templates. • If an Amazon RDS is present, retrieve a list of the available DB engines and information about provisioned RDS instances. For more information, see The managed-updates service-linked role.	August 23, 2022

Change	Description	Date
AWSElasticBeanstalkReadOnlyAccess – Deprecated GovCloud (US) AWS Region	This policy has been replaced by `AWSElasticBeanstalkReadOnly` . This policy will be phased out in the GovCloud (US) AWS Region. When this policy is phased out, it will no longer be available for attachment to new IAM users, groups, or roles after June 17, 2021. For more information, see User policies.	June 17, 2021
AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy –Updated existing policy	This policy was updated to allow Elastic Beanstalk to read attributes for EC2 Availability Zones. It enables Elastic Beanstalk to provide more effective validation of your instance type selection across Availability Zones. For more information, see Managed service role policies.	June 16, 2021

Change	Description	Date
AWSElasticBeanstalkFullAccess – Deprecated GovCloud (US) AWS Region	This policy has been replaced by `AdministratorAccess-AWSElasticBeanstalk`. This policy will be phased out in the GovCloud (US) AWS Region. When this policy is phased out, it will no longer be available for attachment to new IAM users, groups, or roles after June 10, 2021. For more information, see User policies.	June 10, 2021

Change	Description	Date
The following managed policies were deprecated in all of the China AWS Regions: • **AWSElasticBeanstalkFullAccess** • **AWSElasticBeanstalkReadOnlyAccess**	The `AWSElasticBeanstalkFullAccess` policy has been replaced by `AdministratorAccess-AWSElasticBeanstalk` . The `AWSElasticBeanstalkReadOnlyAccess` policy has been replaced by `AWSElasticBeanstalkReadOnly` . These policies were phased out in all of the China AWS Regions. These policies will no longer be available for attachment to new IAM users, groups, or roles after June 3, 2021. For more information, see User policies.	June 3, 2021

Change	Description	Date
AWSElasticBeanstalkService – Deprecated	This policy has been superseded by `AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy` . This policy is phased out and is no longer available for attachment to new IAM users, groups, or roles. For more information, see Managed service role policies.	June 2021 - January 2022

Change	Description	Date
The following managed policies were deprecated in all AWS Regions, except for China and GovCloud (US): • **AWSElasticBeanstalkFullAccess** • **AWSElasticBeanstalkReadOnlyAccess**	The `AWSElasticBeanstalkFullAccess` policy has been replaced by `AdministratorAccess-AWSElasticBeanstalk` . The `AWSElasticBeanstalkReadOnlyAccess` policy has been replaced by `AWSElasticBeanstalkReadOnly` . These policies were phased out in all the AWS Regions, except for China and GovCloud (US). These policies will no longer be available for attachment to new IAM users, groups, or roles after April 16, 2021. For more information, see User policies.	April 16, 2021

Change	Description	Date
The following managed policies were updated: • **AdministratorAccess-AWSElasticBeanstalk** • **AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy**	Both of these policies now support PassRole permissions in China AWS Regions. For more information about `AdministratorAccess-AWSElasticBeanstalk` , see User policies. For more information about `AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy` , see Managed service role policies.	March 9, 2021
AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy – New policy	Elastic Beanstalk added a new policy to replace the `AWSElasticBeanstalkService` managed policy. This new managed policy improves security for your resources by applying a more restrictive set of permissions. For more information, see Managed service role policies.	March 3, 2021
Elastic Beanstalk started tracking changes	Elastic Beanstalk started tracking changes for AWS managed policies.	March 1, 2021

Logging and monitoring in Elastic Beanstalk

AWS provides several tools for monitoring your Elastic Beanstalk resources and responding to potential incidents. Monitoring is important for maintaining the reliability, availability, and performance of AWS Elastic Beanstalk and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multipoint failure if one occurs.

For more information about monitoring, see Monitoring environments in Elastic Beanstalk.

For other Elastic Beanstalk security topics, see AWS Elastic Beanstalk security.

Enhanced health reporting

Enhanced health reporting is a feature that you can enable on your environment to allow Elastic Beanstalk to gather additional information about resources in your environment. Elastic Beanstalk analyzes the information to provide a better picture of overall environment health and help identify issues that can cause your application to become unavailable. For more information, see Enhanced health reporting and monitoring in Elastic Beanstalk.

Amazon EC2 instance logs

The Amazon EC2 instances in your Elastic Beanstalk environment generate logs that you can view to troubleshoot issues with your application or configuration files. Logs created by the web server, application server, Elastic Beanstalk platform scripts, and AWS CloudFormation are stored locally on individual instances. You can easily retrieve them by using the environment management console or the EB CLI. You can also configure your environment to stream logs to Amazon CloudWatch Logs in real time. For more information, see Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment.

Environment notifications

You can configure your Elastic Beanstalk environment to use Amazon Simple Notification Service (Amazon SNS) to notify you of important events that affect your application. Specify an email address during or after environment creation to receive emails from AWS when an error occurs, or when your environment's health changes. For more information, see Elastic Beanstalk environment notifications with Amazon SNS.

Amazon CloudWatch alarms

Using CloudWatch alarms, you watch a single metric over a time period that you specify. If the metric exceeds a given threshold, a notification is sent to an Amazon SNS topic or AWS Auto Scaling policy. CloudWatch alarms don't invoke actions because they are in a particular state.

Instead, alarms invoke actions when the state changed and was maintained for a specified number of periods. For more information, see Using Elastic Beanstalk with Amazon CloudWatch.

AWS CloudTrail logs

CloudTrail provides a record of actions taken by a user, role, or an AWS service in Elastic Beanstalk. Using the information collected by CloudTrail, you can determine the request that was made to Elastic Beanstalk, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, see Logging Elastic Beanstalk API calls with AWS CloudTrail.

AWS X-Ray debugging

X-Ray is an AWS service that gathers data about the requests that your application serves, and uses it to construct a service map that you can use to identify issues with your application and opportunities for optimization. You can use the AWS Elastic Beanstalk console or a configuration file to run the X-Ray daemon on the instances in your environment. For more information, see Configuring AWS X-Ray debugging.

Compliance validation for Elastic Beanstalk

The security and compliance of AWS Elastic Beanstalk is assessed by third-party auditors as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others. AWS provides a frequently updated list of AWS services in scope of specific compliance programs at AWS Services in Scope by Compliance Program.

Third-party audit reports are available for you to download using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

For more information about AWS compliance programs, see AWS Compliance Programs.

Your compliance responsibility when using Elastic Beanstalk is determined by the sensitivity of your data, your organization's compliance objectives, and applicable laws and regulations. If your use of Elastic Beanstalk is subject to compliance with standards such as HIPAA, PCI, or FedRAMP, AWS provides resources to help:

- Security and Compliance Quick Start Guides – Deployment guides that discuss architectural considerations and provide steps for deploying security-focused and compliance-focused baseline environments on AWS.

- Architecting for HIPAA Security and Compliance on Amazon Web Services – A whitepaper that describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – A collection of compliance workbooks and guides that might apply to your industry and location.
- AWS Config – A service that assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – A comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

For other Elastic Beanstalk security topics, see AWS Elastic Beanstalk security.

Resilience in Elastic Beanstalk

AWS Elastic Beanstalk manages and automates the use of the AWS global infrastructure on your behalf. When using Elastic Beanstalk, you benefit from the availability and fault tolerance mechanisms that AWS offers.

The AWS global infrastructure is built around AWS Regions and Availability Zones.

AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking.

With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

For other Elastic Beanstalk security topics, see AWS Elastic Beanstalk security.

Infrastructure security in Elastic Beanstalk

As a managed service, AWS Elastic Beanstalk is protected by the AWS global network security procedures that are described in our Best Practices for Security, Identity, and Compliance website.

You use AWS published API calls to access Elastic Beanstalk through the network. Clients must support Transport Layer Security (TLS) 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS), such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern platforms support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

For other Elastic Beanstalk security topics, see AWS Elastic Beanstalk security.

Configuration and vulnerability analysis in Elastic Beanstalk

AWS and our customers share responsibility for achieving a high level of software component security and compliance. AWS Elastic Beanstalk helps you perform your side of the shared responsibility model by providing a *managed updates* feature. This feature automatically applies patch and minor updates for an Elastic Beanstalk supported platform version.

For more information, see Shared responsibility model for Elastic Beanstalk platform maintenance.

For other Elastic Beanstalk security topics, see AWS Elastic Beanstalk security.

Security best practices for Elastic Beanstalk

AWS Elastic Beanstalk provides several security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don’t represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations, not prescriptions.

For other Elastic Beanstalk security topics, see AWS Elastic Beanstalk security.

Preventive security best practices

Preventive security controls attempt to prevent incidents before they occur.

Implement least privilege access

Elastic Beanstalk provides AWS Identity and Access Management (IAM) managed policies for instance profiles, service roles, and IAM users. These managed policies specify all permissions that might be necessary for the correct operation of your environment and application.

Your application might not require all the permissions in our managed policies. You can customize them and grant only the permissions that are required for your environment's instances, the Elastic Beanstalk service, and your users to perform their tasks. This is particularly relevant to user policies, where different user roles might have different permission needs. Implementing least privilege access is fundamental in reducing security risk and the impact that could result from errors or malicious intent.

Protect sensitive application data

When your application needs to access sensitive information like credentials, API keys, or configuration data, follow these practices to maintain security:

- Retrieve sensitive data directly from AWS Secrets Manager or AWS Systems Manager Parameter Store using their respective SDKs or APIs in your application code. This provides the most secure and flexible way to access sensitive information.
- If you pass sensitive data from AWS Secrets Manager or AWS Systems Manager Parameter Store as environment variables (see Fetch secrets to environment variables), carefully restrict access to EC2 key pairs and configure appropriate IAM roles with least-privilege permissions for your instances.
- Never print, log, or expose sensitive data in your application code, as these values could end up in log files or error messages that might be visible to unauthorized users.

Update your platforms regularly

Elastic Beanstalk regularly releases new platform versions to update all of its platforms. New platform versions provide operating system, runtime, application server, and web server updates, and updates to Elastic Beanstalk components. Many of these platform updates include important security fixes. Ensure that your Elastic Beanstalk environments are running on a supported platform version (typically the latest version for your platform). For details, see Updating your Elastic Beanstalk environment's platform version.

The easiest way to keep your environment's platform up to date is to configure the environment to use managed platform updates.

Enforce IMDSv2 on environment instances

Amazon Elastic Compute Cloud (Amazon EC2) instances in your Elastic Beanstalk environments use the instance metadata service (IMDS), an on-instance component, to securely access instance metadata. IMDS supports two methods for accessing data: IMDSv1 and IMDSv2. IMDSv2 uses session-oriented requests and mitigates several types of vulnerabilities that could be used to try to access the IMDS. For details about the advantages of IMDSv2, see enhancements to add defense in depth to the EC2 Instance Metadata Service.

IMDSv2 is more secure, so it's a good idea to enforce the use of IMDSv2 on your instances. To enforce IMDSv2, ensure that all components of your application support IMDSv2, and then disable IMDSv1. For more information, see the section called “IMDS”.

Detective security best practices

Detective security controls identify security violations after they have occurred. They can help you detect a potential security threat or incident.

Implement monitoring

Monitoring is an important part of maintaining the reliability, security, availability, and performance of your Elastic Beanstalk solutions. AWS provides several tools and services to help you monitor your AWS services.

The following are some examples of items to monitor:

- *Amazon CloudWatch metrics for Elastic Beanstalk* – Set alarms for key Elastic Beanstalk metrics and for your application's custom metrics. For details, see Using Elastic Beanstalk with Amazon CloudWatch.
- *AWS CloudTrail entries* – Track actions that might impact availability, like `UpdateEnvironment` or `TerminateEnvironment`. For details, see Logging Elastic Beanstalk API calls with AWS CloudTrail.

Enable AWS Config

AWS Config provides a detailed view of the configuration of AWS resources in your account. You can see how resources are related, get a history of configuration changes, and see how relationships and configurations change over time.

You can use AWS Config to define rules that evaluate resource configurations for data compliance. AWS Config rules represent the ideal configuration settings for your Elastic Beanstalk resources. If a resource violates a rule and is flagged as *noncompliant*, AWS Config can alert you using an

Amazon Simple Notification Service (Amazon SNS) topic. For details, see Finding and tracking Elastic Beanstalk resources with AWS Config.

Elastic Beanstalk Service roles, instance profiles, and user policies

Roles are an entities that you create with AWS Identity and Access Management (IAM) to apply permissions. There are required roles for your Elastic Beanstalk environment to function properly. You also have the option to create your own custom policies and roles that you can assign to users or groups.

Required roles for your Elastic Beanstalk environment

When you create an environment, AWS Elastic Beanstalk prompts you to provide the following AWS Identity and Access Management (IAM) roles:

- Service role: Elastic Beanstalk assumes a service role to use other AWS services on your behalf.
- Instance profile Elastic Beanstalk applies an instance profile to the Amazon EC2 instances in your environment. This action allows them to perform required tasks, such as retrieving information from Amazon Simple Storage Service (Amazon S3) and uploading logs to S3.

Create the service role and EC2 instance profile role

If your AWS account doesn't have an EC2 instance profile or a service role, you must create one of each using the IAM service. You can then assign the EC2 instance profile and service role to new environments that you create. The **Create environment** wizard guides you to the IAM service, so that you can create these roles with the required permissions.

Optional polices and roles to manage your Elastic Beanstalk environment

You can optionally create user policies and apply them to IAM users and groups in your account. Doing so allows the users to create and manage Elastic Beanstalk applications and environments. You can also assign Elastic Beanstalk managed policies for full access and read-only access to users or groups. For more information about these policies, see the section called "User policies".

You can create your own instance profiles and user policies for advanced scenarios. If your instances need to access services that aren't included in the default policies, you can create a new policy or add additional policies to the default one. If the managed policy is too permissive for your needs, you can also create more restrictive user policies. For more information about AWS permissions, see the IAM User Guide.

Elastic Beanstalk service role

A service role is the IAM role that Elastic Beanstalk assumes when calling other services on your behalf. For example, Elastic Beanstalk uses a service role when it calls Amazon Elastic Compute Cloud (Amazon EC2), Elastic Load Balancing, and Amazon EC2 Auto Scaling APIs to gather information. The service role that Elastic Beanstalk uses is the one that you specified when you create the Elastic Beanstalk environment.

There are two managed policies that are attached to the service role. These policies provide the permissions that allow Elastic Beanstalk to access the required AWS resources to create and manage your environments. One managed policy provides permissions for enhanced health monitoring and worker tier Amazon SQS support, and another one provides additional permissions required for managed platform updates.

`AWSElasticBeanstalkEnhancedHealth`

This policy grants permissions for Elastic Beanstalk to monitor instance and environment health. It also includes Amazon SQS actions to allow Elastic Beanstalk to monitor queue activity for worker environments. To view the content of this managed policy, see the AWSElasticBeanstalkEnhancedHealth page in the *AWS Managed Policy Reference Guide*.

`AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy`

This policy grants permissions for Elastic Beanstalk to update environments on your behalf to perform managed platform updates. To view the content of this managed policy, see the AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy page in the *AWS Managed Policy Reference Guide*.

Service-level permission groupings

This policy is grouped into statements based on the set of permissions provided.

- *`ElasticBeanstalkPermissions`* – This group of permissions is for calling the Elastic Beanstalk service actions (Elastic Beanstalk APIs).

- *AllowPassRoleToElasticBeanstalkAndDownstreamServices* – This group of permissions allows any role to be passed to Elastic Beanstalk and to other downstream services like AWS CloudFormation.
- *ReadOnlyPermissions* – This group of permissions is for collecting information about the running environment.
- **OperationPermissions* – Groups with this naming pattern are for calling the necessary operations to perform platform updates.
- **BroadOperationPermissions* – Groups with this naming pattern are for calling the necessary operations to perform platform updates. They also include broad permissions for supporting legacy environments.
- **TagResource* – Groups with this naming pattern are for calls that use the tag-on-create APIs to attach tags on resources that are being created in an Elastic Beanstalk environment.

You may create an Elastic Beanstalk environment with any of the following approaches. Each section describes how the approach handles the service role.

Elastic Beanstalk console

If you create an environment using the Elastic Beanstalk console, Elastic Beanstalk prompts you to create a service role that's named `aws-elasticbeanstalk-service-role`. When created via Elastic Beanstalk, this role includes a trust policy that allows Elastic Beanstalk to assume the service role. The two managed policies described earlier in this topic are also attached to the role.

Elastic Beanstalk Command Line Interface (EB CLI)

You may create an environment using the the section called "**eb create**" command of the Elastic Beanstalk Command Line Interface (EB CLI). If you don't specify a service role through the `--service-role` option. Elastic Beanstalk creates the same default service role `aws-elasticbeanstalk-service-role`. If the default service role already exists, Elastic Beanstalk uses it for the new environment. When created via Elastic Beanstalk, this role includes a trust policy that allows Elastic Beanstalk to assume the service role. The two managed policies described earlier in this topic are also attached to the role.

Elastic Beanstalk API

You may create an environment using the `CreateEnvironment` action of the Elastic Beanstalk API. If you don't specify a service role, Elastic Beanstalk creates a monitoring service-linked role.

This is a unique type of service role that is predefined by Elastic Beanstalk to include all the permissions that the service requires to call other AWS services on your behalf. The service-linked role is associated with your account. Elastic Beanstalk creates it once, and then reuses it when creating additional environments. You can also use IAM to create the monitoring service-linked role for your account in advance. When your account has a monitoring service-linked role, you can use it to create an environment using either the Elastic Beanstalk console, the Elastic Beanstalk API, or the EB CLI. For instructions on how to use service-linked roles with Elastic Beanstalk environments, see Using service-linked roles for Elastic Beanstalk.

For more information about service roles, see Managing Elastic Beanstalk service roles.

Elastic Beanstalk instance profile

An instance profile is an IAM role that's applied to Amazon EC2 instances that are launched in your Elastic Beanstalk environment. When creating an Elastic Beanstalk environment, you specify the instance profile that's used when your EC2 instances take the following actions:

- Retrieve application versions from Amazon Simple Storage Service (Amazon S3)
- Write logs to Amazon S3
- In AWS X-Ray integrated environments, upload debugging data to X-Ray
- In Amazon ECS managed Docker environments, coordinate container deployments with Amazon Elastic Container Service (Amazon ECS)
- In worker environments, read from an Amazon Simple Queue Service (Amazon SQS) queue
- In worker environments, perform leader election with Amazon DynamoDB
- In worker environments, publish instance health metrics to Amazon CloudWatch

Managed policies

Elastic Beanstalk provides a set of managed policies that allow the EC2 instances in your environment to perform required operations. The managed policies required for basic use cases are the following.

- `AWSElasticBeanstalkWebTier`
- `AWSElasticBeanstalkWorkerTier`
- `AWSElasticBeanstalkMulticontainerDocker`

If your web application requires access to other additional AWS services, add statements or managed policies to the instance profile that allow access to those services. For more information, see Adding permissions to the default instance profile.

Creating an EC2 instance profile

If your AWS account doesn't have an EC2 instance profile, you must create one using the IAM service. You can then assign the EC2 instance profile to new environments that you create. The **Create environment** steps in the Elastic Beanstalk console provides you access to the IAM console, so that you can create an EC2 instance profile with the required permissions.

You can also create an EC2 instance profile by directly accessing the IAM console, without going through the Elastic Beanstalk console. For detailed steps to create an Elastic Beanstalk EC2 instance profile in the IAM console, see Creating an instance profile.

Elastic Beanstalk user policy

Create IAM users for each user who uses Elastic Beanstalk to avoid using your root account or sharing credentials. As a security best practice, only grant these users permissions to access services and features that they need.

Elastic Beanstalk requires permissions not only for its own API actions, but also for several other AWS services. Elastic Beanstalk uses user permissions to launch resources in an environment. These resources include EC2 instances, an Elastic Load Balancing load balancer, and an Auto Scaling group. Elastic Beanstalk also uses user permissions to save logs and templates to Amazon Simple Storage Service (Amazon S3), send notifications to Amazon SNS, assign instance profiles, and publish metrics to CloudWatch. Elastic Beanstalk requires AWS CloudFormation permissions to orchestrate resource deployments and updates. It also requires Amazon RDS permissions to create databases when needed, and Amazon SQS permissions to create queues for worker environments.

For more information about user policies, see Managing Elastic Beanstalk user policies.

Tutorials and samples

Language and framework specific tutorials are spread throughout the AWS Elastic Beanstalk Developer Guide. New and updated tutorials are added to this list as they are published. The most recent updates are shown first.

These tutorials are targeted at intermediate users and may not contain instructions for basic steps such as signing up for AWS. If this is your first time using AWS or Elastic Beanstalk, check out the Getting Started walkthrough to get your first Elastic Beanstalk environment up and running.

- **Ruby on Rails** - Deploying a rails application to Elastic Beanstalk
- **Ruby and Sinatra** - Deploying a sinatra application to Elastic Beanstalk
- **PHP and MySQL HA Configuration** - Deploying a high-availability PHP application with an external Amazon RDS database to Elastic Beanstalk
- **PHP and Laravel** - Deploying a Laravel application to Elastic Beanstalk
- **PHP and CakePHP** - Deploying a CakePHP application to Elastic Beanstalk
- **PHP and Drupal HA Configuration** - Deploying a high-availability Drupal website with an external Amazon RDS database to Elastic Beanstalk
- **PHP and WordPress HA Configuration** - Deploying a high-availability WordPress website with an external Amazon RDS database to Elastic Beanstalk
- **Node.js with DynamoDB HA Configuration** - Deploying a Node.js application with DynamoDB to Elastic Beanstalk
- **ASP.NET Core** - QuickStart: Deploy an ASP.NET application to Elastic Beanstalk
- **Python and Flask** - Deploying a Flask application to Elastic Beanstalk
- **Python and Django** - Deploying a Django application to Elastic Beanstalk
- **Node.js and Express** - Deploying a Node.js Express application to Elastic Beanstalk
- **Docker, PHP and nginx** - Creating an ECS managed Docker environment with the Elastic Beanstalk console

You can download the sample applications used by Elastic Beanstalk when you create an environment without providing a source bundle with the following links:

- **Docker** – docker.zip

- **Multicontainer Docker** – docker-multicontainer-v2.zip
- **Preconfigured Docker (Glassfish)** – docker-glassfish-v1.zip
- **Go** – go.zip
- **Corretto** – corretto.zip
- **Tomcat** – tomcat.zip
- **.NET Core on Linux** – dotnet-core-linux.zip
- **.NET Core** – dotnet-asp-windows.zip
- **Node.js** – nodejs.zip
- **PHP** – php.zip
- **Python** – python.zip
- **Ruby** – ruby.zip

More involved sample applications that show the use of additional web frameworks, libraries and tools are available as open source projects on GitHub:

- **Load-balanced WordPress** (tutorial) – Configuration files for installing WordPress securely and running it in a load-balanced Elastic Beanstalk environment.
- **Load-balanced Drupal** (tutorial) – Configuration files and instructions for installing Drupal securely and running it in a load-balanced Elastic Beanstalk environment.
- **Scorekeep** - RESTful web API that uses the Spring framework and the AWS SDK for Java to provide an interface for creating and managing users, sessions, and games. The API is bundled with an Angular 1.5 web app that consumes the API over HTTP. Includes branches that show integration with Amazon Cognito, AWS X-Ray, and Amazon Relational Database Service.

  The application uses features of the Java SE platform to download dependencies and build on-instance, minimizing the size of the souce bundle. The application also includes nginx configuration files that override the default configuration to serve the frontend web app statically on port 80 through the proxy, and route requests to paths under `/api` to the API running on `localhost:5000`.
- **Does it Have Snakes?** - Tomcat application that shows the use of RDS in a Java EE web application in Elastic Beanstalk. The project shows the use of Servlets, JSPs, Simple Tag Support, Tag Files, JDBC, SQL, Log4J, Bootstrap, Jackson, and Elastic Beanstalk configuration files.
- **Locust Load Generator** - This project shows the use of Java SE platform features to install and run Locust, a load generating tool written in Python. The project includes configuration files that

install and configure Locust, a build script that configures a DynamoDB table, and a Procfile that runs Locust.

- **Share Your Thoughts** (tutorial) - PHP application that shows the use of MySQL on Amazon RDS, Composer, and configuration files.
- **A New Startup** (tutorial) - Node.js sample application that shows the use of DynamoDB, the AWS SDK for JavaScript in Node.js, npm package management, and configuration files.

Migrating IIS applications to Elastic Beanstalk

AWS Elastic Beanstalk provides a streamlined migration path for your Windows applications running on Internet Information Services (IIS). The migration capability described in this chapter significantly reduces the time and complexity that's typically associated with cloud migrations, helping you to maintain application functionality and configuration integrity during the transition to AWS.

The eb migrate operation

Use the **eb migrate** command in the Elastic Beanstalk Command Line Interface (EB CLI), to automatically discover, package, and deploy your IIS applications to the AWS Cloud. The process maintains application functionality and preserves your configurations, including bindings, application pools, and authentication settings.

The following steps summarize the process that the `eb migrate` operation performs to transition your application to the AWS Cloud:

1. Discover IIS sites and their configurations.
2. Package application content and configuration.
3. Create Elastic Beanstalk environment and application.
4. Deploy the application with preserved settings.

Workflow and location execution options

The **eb migrate** command provides options for flexible migration workflows and execution locations. By default, run the command on the target server that contains the application you want to migrate to Elastic Beanstalk. If you can't run commands directly on the application server, use the `remote` option to run the command from a bastion host that connects to the target server containing your application and configurations. To complete the migration in two steps, you can also generate the migration package without deploying it using the `archive-only` option and then deploy it later at your convenience using the `archive` option.

For reference information about the **eb migrate** command, see the section called "**eb migrate**".

Topics

The following topics provide detailed information about migrating IIS applications to Elastic Beanstalk:

- the section called "Prerequisites" - Understand the required software, access, and permissions to migrate your Windows applications to AWS Elastic Beanstalk environments.
- the section called "Migration glossary" - Understand how IIS components map to Elastic Beanstalk resources
- the section called "IIS to Elastic Beanstalk migration mapping" - Understand how IIS components map to Elastic Beanstalk resources
- the section called "Basic IIS migrations" - Learn how to perform basic migrations
- the section called "Advanced migration scenarios" - Handle complex migration scenarios
- the section called "Security configuration" - Configure security settings during migration
- the section called "Network configuration" - Manage network and port configurations
- the section called "Troubleshooting and diagnostics" - Troubleshoot common migration issues
- the section called "Migration options: EB CLI vs. MGN" - Compare two primary migration options.

Prerequisites

Before using the **eb migrate** command, ensure your environment meets these requirements:

IIS installation and version

The server that you're migrating from must run Internet Information Services (IIS) version 7.0 or later. IIS 10.0 on Windows Server 2016 or later provides the most compatible environment for migration.

To verify your IIS version run the following command:

```
PS C:\migrations_workspace> Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\InetStp\"
...
SetupString                : IIS 10.0
VersionString              : Version 10.0
...
```

Windows Server requirements

Your source environment should run Windows Server 2016 or later for optimal compatibility. Elastic Beanstalk supports these Windows Server versions as target platforms:

- Windows Server 2025
- Windows Server 2022
- Windows Server 2019
- Windows Server 2016

EB CLI installation

- *Default workflow (without the `--remote` option)*:
  - Python and the Elastic Beanstalk Command Line Interface (EB CLI) must be installed on the server that contains the application that you want to migrate to Elastic Beanstalk. While it is not required, we recommend installing the EB CLI inside a `virtualenv` sandbox as described in the section called "Install in virtualenv".
- *Using the `--remote` option*:
  - Python and the Elastic Beanstalk Command Line Interface (EB CLI) must be installed on your bastion host. While it is not required, we recommend installing the EB CLI inside a `virtualenv` sandbox as described in the section called "Install in virtualenv".

Required permissions

You need the following credentials and permissions:

- Administrator privileges on the source IIS server or on the bastion host (if using the `--remote` option).
- AWS credentials with permissions to create and manage Elastic Beanstalk resources

Web Deploy 3.6

Microsoft Web Deploy tool (version 3.6 or later) must be installed on your source server or on the bastion host (if using the `--remote` option). This tool is used by **eb migrate** to package your applications.

To verify the installation run the following command:

:

```
PS C:\migrations_workspace> Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\IIS
 Extensions\MSDeploy\3" -Name InstallPath

InstallPath  : C:\Program Files\IIS\Microsoft Web Deploy V3\
...
```

For installation instructions, see Installing and Configuring Web Deploy on IIS 8.0 or Later on the Microsoft Windows product documentation website.

Network requirements

- *Default workflow (without the `--remote` option)*:
  - Your source server must have outbound internet access to AWS services.
- *Using the `--remote` option*:
  - Your source server must have outbound internet access to AWS services.
  - Configure the proper security group ingress rules that allow for an outgoing network connection from your bastion host and an incoming connection into the remote machine. Ensure the IP of the bastion host is allow-listed via TCP on port 22 to access the remote machine.
  - Ensure your SSH client is installed and running on your remote machine as well as your bastion host.
  - Ensure that your firewall configuration contains the appropriate rules that open up port 22 or allow connection to the client.
  - Test your connection by manually SSH-ing into the remote host from the bastion host before attempting migration.

Migration glossary

This glossary provides definitions for key terms and concepts related to IIS, Elastic Beanstalk, and the migration of IIS applications to Elastic Beanstalk.

Windows, IIS, and .NET terms

IIS

Internet Information Services, a web server software developed by Microsoft for use with Windows Server. IIS hosts websites, web applications, and web services, providing a platform for running ASP.NET and other web technologies. During migration to Elastic Beanstalk, IIS sites and their configurations are packaged and deployed to Windows Server instances in the AWS Cloud.

IIS versions 7.0 and later are supported for migration, with IIS 10.0 on Windows Server 2016 or later providing the most compatible environment.

.NET Framework

A software development platform developed by Microsoft for building and running Windows applications. It provides a large class library called Framework Class Library (FCL) and supports language interoperability across several programming languages.

When migrating to Elastic Beanstalk, applications built on the .NET Framework continue to run on the same version of the framework in the cloud environment. Elastic Beanstalk supports multiple .NET Framework versions (4.x) on its Windows Server platforms.

.NET Core

A cross-platform, open-source successor to the .NET Framework, designed to be more modular and lightweight. .NET Core (now simply called .NET 5 and later) enables developers to build applications that run on Windows, Linux, and macOS.

When migrating applications built on .NET Core to Elastic Beanstalk, you can choose between Windows Server platforms or Linux-based platforms, depending on your application's requirements and dependencies.

Common Language Runtime (CLR)

The virtual machine component of the .NET Framework that manages the execution of .NET programs. The CLR provides services such as memory management, type safety, exception handling, garbage collection, and thread management.

When migrating to Elastic Beanstalk, the appropriate CLR version is automatically available on the Windows Server platform you select, ensuring compatibility with your application's requirements.

Site

A logical container in IIS that represents a web application or service, identified by a unique binding of IP address, port, and host header. Each IIS site has its own application pool, bindings, and configuration settings, and can contain one or more applications.

Application

A grouping of content and code files within an IIS site that handles requests for a specific URL space. Applications in IIS can have their own configuration settings, which are typically stored in `web.config` files.

When migrating to Elastic Beanstalk, applications are preserved with their path structure and configuration settings. The migration process ensures that nested applications maintain their hierarchy and URL paths in the cloud environment.

ApplicationPool

An IIS feature that isolates web applications for better security, reliability, and performance management. Application pools define the runtime environment for applications, including the .NET Framework version, pipeline mode, and identity settings.

VirtualDirectory

A directory mapping in IIS that allows content to be served from a location outside the site's root directory. Virtual directories enable you to organize content across different physical locations while presenting a unified URL structure to users.

When migrating to Elastic Beanstalk, virtual directories are preserved with their path mappings. The **eb migrate** command creates the necessary directory structure and configurations in the cloud environment to maintain the same URL paths.

ARR

Application Request Routing, an IIS extension that provides load balancing and proxying capabilities for web servers. ARR enables URL-based routing, HTTP request forwarding, and load distribution across multiple servers.

During migration to Elastic Beanstalk, ARR configurations are preserved through the installation of ARR features on EC2 instances and the configuration of appropriate routing rules. For complex routing scenarios, the migration process may also leverage Application Load Balancer rules to implement similar functionality.

URL Rewrite

An IIS module that modifies requested URLs based on defined rules before they reach the web application. URL Rewrite enables URL manipulation, redirection, and content delivery based on patterns and conditions.

When migrating to Elastic Beanstalk, URL rewrite rules from your `web.config` files are translated into ALB routing rules where possible, or preserved in the IIS configuration on the EC2 instances. This ensures that URL patterns and redirects continue to function as expected in the cloud environment.

msdeploy.exe

A command-line tool used for deploying web applications and websites to IIS servers. Also known as Web Deploy, it provides a way to package, synchronize, and deploy web applications, websites, and server configurations.

The **eb migrate** command uses Web Deploy (version 3.6 or later) to package your applications during migration to Elastic Beanstalk. This tool must be installed on your source server for the migration process to work correctly.

Physical Path

The actual file system location where an IIS site or application's content files are stored. Physical paths can point to local directories, network shares, or other storage locations accessible to the IIS server.

During migration to Elastic Beanstalk, physical paths are mapped to appropriate locations on the EC2 instances in your environment. The migration process preserves the content structure while ensuring that all files are properly deployed to the cloud environment.

applicationHost.config

The root configuration file for IIS that defines server-wide settings and contains configuration for all sites, applications, and virtual directories. This file is located in the `%windir%\System32\inetsrv\config` directory and controls the overall behavior of the IIS server.

When migrating to Elastic Beanstalk, relevant settings from `applicationHost.config` are extracted and applied to the IIS configuration on the EC2 instances in your environment. This ensures that server-wide settings are preserved during migration.

web.config

An XML-based configuration file used in ASP.NET applications to control application settings, security, and behavior at the application or directory level. `web.config` files can contain settings for authentication, authorization, session state, compilation, and custom application parameters.

During migration to Elastic Beanstalk, `web.config` files are preserved and deployed with your application. The migration process ensures that application-specific configurations continue to function as expected in the cloud environment.

DefaultDocument

An IIS feature that specifies the default file to serve when a user requests a directory without specifying a file name. Default documents are enabled by default, and IIS 7 defines the following default document files in the `applicationHost.config` file as server-wide defaults: Default.htm, Default.asp, Index.htm, Index.html, Iisstart.htm.

When migrating to Elastic Beanstalk, default document settings are preserved in the IIS configuration on the EC2 instances, ensuring that directory requests are handled consistently in the cloud environment.

system.webServer

A configuration section in `web.config` or `applicationHost.config` that contains IIS-specific settings for modules, handlers, and other server behaviors. This section controls how IIS processes requests, manages modules, and configures server features.

During migration to Elastic Beanstalk, system.webServer configurations are preserved in your application's `web.config` files and applied to the IIS installation on the EC2 instances in your environment. This ensures that IIS-specific behaviors are maintained in the cloud environment.

Elastic Beanstalk terms

Platform

A combination of operating system, programming language runtime, web server, application server, and Elastic Beanstalk components that define the software stack for running applications.

For Windows migrations, Elastic Beanstalk provides platforms based on Windows Server 2016, 2019, and 2022 with IIS and various .NET Framework versions to ensure compatibility with your source environment.

SolutionStack

A predefined platform configuration in Elastic Beanstalk that specifies the operating system, runtime, and other components needed to run an application. Conceptually identical to a platform and used interchangeably to operate environments.

During migration, the **eb migrate** command selects an appropriate solution stack based on your source environment's configuration, ensuring compatibility with your IIS applications.

CreateEnvironment

An Elastic Beanstalk API action that creates a new environment to host an application version. This API is used by the **eb migrate** command to provision the necessary AWS resources for your migrated application.

The migration process configures appropriate environment parameters based on your source IIS environment, including instance type, environment variables, and option settings.

CreateApplicationVersion

An Elastic Beanstalk API action that creates a new application version from a source bundle stored in Amazon S3. The **eb migrate** command uses this API to register your packaged IIS application as a version in Elastic Beanstalk.

During migration, your application files and configuration are packaged, uploaded to Amazon S3, and registered as an application version before deployment.

DescribeEvents

An Elastic Beanstalk API action that retrieves a list of events for an environment, including deployments, configuration changes, and operational issues. The **eb migrate** command uses this API to monitor the progress of your migration.

You can also use the **eb events** command after migration to view the history of events for your environment.

DescribeEnvironmentHealth

An Elastic Beanstalk API action that provides detailed health information about the instances and other components of an environment. This API is used to verify the health of your migrated application after deployment.

After migration, you can use the **eb health** command to check the status of your environment and identify any issues that need attention.

HealthD

A monitoring agent in Elastic Beanstalk that collects metrics, monitors logs, and reports health status for EC2 instances in an environment. HealthD provides enhanced health reporting for your migrated applications.

After migration, HealthD monitors your application's performance, resource utilization, and request success rates, providing a comprehensive view of your environment's health.

Bundle Logs

A feature in Elastic Beanstalk that compresses and uploads logs from EC2 instances to Amazon S3 for centralized storage and analysis. This feature helps you troubleshoot issues with your migrated applications.

After migration, you can use the **eb logs** command to retrieve and view logs from your environment.

aws-windows-deployment-manifest.json

A file that describes the contents, dependencies, and configuration of a software package or application. This manifest is generated during the migration process to define how your IIS applications should be deployed on Elastic Beanstalk.

custom manifest section

A section within `aws-windows-deployment-manifest.json` that provides custom control over application deployment. This section contains PowerShell scripts and commands that are executed during the deployment process.

During migration, custom manifest sections are generated to handle specific aspects of your IIS configuration, such as virtual directory setup, permission management, and application pool configuration.

EB CLI

A command-line tool that provides commands for creating, configuring, and managing Elastic Beanstalk applications and environments. The EB CLI includes the **eb migrate** command specifically for migrating IIS applications to Elastic Beanstalk.

After migration, you can continue to use the EB CLI to manage your environment, deploy updates, monitor health, and perform other administrative tasks.

Option settings

Configuration values that define how Elastic Beanstalk provisions and configures AWS resources in your environment. Option settings are organized into namespaces that represent different components of your environment, such as load balancers, instances, and environment processes.

During migration, the **eb migrate** command generates appropriate option settings based on your IIS configuration to ensure that your cloud environment matches your source environment's capabilities. For more information, see Configuration options in the Elastic Beanstalk Developer Guide.

aws:elbv2:listener:default

An Elastic Beanstalk configuration namespace for the default listener on an Application Load Balancer. During migration, this namespace is configured based on your IIS site bindings to ensure proper traffic routing.

The default listener typically handles HTTP traffic on port 80, which is then forwarded to your application instances according to the routing rules.

aws:elbv2:listener:listener_port

An Elastic Beanstalk configuration namespace for a specific listener port on an Application Load Balancer. This namespace is used to configure additional listeners for your migrated applications, such as HTTPS on port 443.

During migration, listeners are created based on the port bindings of your IIS sites, ensuring that your applications remain accessible on the same ports as in your source environment.

aws:elbv2:listenerrule:rule_name

An Elastic Beanstalk configuration namespace for defining routing rules for an Application Load Balancer listener. These rules determine how incoming requests are routed to different target groups based on path patterns or host headers.

During migration, listener rules are created to match the URL structure of your IIS applications, ensuring that requests are routed to the correct application paths.

aws:elasticbeanstalk:environment:process:default

An Elastic Beanstalk configuration namespace for the default process in an environment. This namespace defines how the default web application process is configured, including health check settings, port mappings, and proxy settings.

During migration, the default process is configured based on your primary IIS site's settings, ensuring proper health monitoring and request handling.

aws:elasticbeanstalk:environment:process:process_name

An Elastic Beanstalk configuration namespace for a specific named process in an environment. This namespace allows you to define multiple processes with different configurations, similar to having multiple application pools in IIS.

During migration, additional processes may be created to represent different site bindings from your source environment.

Note

For more information about some of the terms described in this topic see the following resources:

- Elastic Beanstalk API actions - AWS Elastic Beanstalk API Reference
- Elastic Beanstalk platforms, including supported platform versions - Supported Platforms in the *AWS Elastic Beanstalk Platforms guide*
- Elastic Beanstalk configuration namespaces - General options for all environments in this guide
- The EB CLI or specific EB CLI commands - Setting up the EB command line interface (EB CLI) to manage Elastic Beanstalk in this guide

Python terms

pip

The package installer for Python, used to install and manage software packages written in Python. The EB CLI is installed and updated using pip.

During the migration process, pip is used to install the EB CLI package and its dependencies on your source server, providing the tools needed for migration.

PyPI

Python Package Index, the official repository for third-party Python software packages, from which pip retrieves and installs packages. The EB CLI and its dependencies are hosted on PyPI.

When installing the EB CLI for migration, pip connects to PyPI to download and install the necessary packages.

virtualenv

A tool to create isolated Python environments, allowing different projects to have their own dependencies and packages without conflicts. Using virtualenv is recommended when installing the EB CLI to avoid conflicts with other Python applications.

Creating a virtual environment before installing the EB CLI ensures that the migration tools have a clean, isolated environment with the correct dependencies.

pywin32

A set of Python extensions that provide access to many of the Windows APIs, enabling interaction with the Windows operating system and its components. The EB CLI uses pywin32 to interact with Windows-specific features during migration.

During the migration process, pywin32 is used to access IIS configuration, Windows registry settings, and other system information needed to properly package and migrate your applications.

pythonnet

A package that enables Python code to interact with .NET Framework and .NET Core applications. This integration allows the EB CLI to work with .NET components during the migration process.

The migration process may use pythonnet to interact with .NET assemblies and components when analyzing and packaging your applications for deployment to Elastic Beanstalk.

Performing basic IIS migrations

This section guides you through the process of migrating your IIS applications to Elastic Beanstalk using the **eb migrate** command.

Exploring your IIS environment

Before making any changes, you'll want to understand what resources exist on your server. Start by exploring your IIS sites by running **eb migrate explore**, as shown in the following example:

```
PS C:\migrations_workspace> eb migrate explore
```

This command reveals your IIS sites. Refer to the following listing:

```
Default Web Site
Intranet
API.Internal
Reports
```

For a detailed view of each site's configuration, including bindings, applications, and virtual directories, add the `--verbose` option, as shown in this example:

```
PS C:\migrations_workspace> eb migrate explore --verbose
```

The following listing shows the comprehensive information about your environment the command provides:

```
1: Default Web Site:
  - Bindings:
    - *:80:www.example.com
    - *:443:www.example.com
  - Application '/':
    - Application Pool: DefaultAppPool
    - Enabled Protocols: http
    - Virtual Directories:
      - /:
        - Physical Path: C:\inetpub\wwwroot
        - Logon Method: ClearText
  - Application '/api':
    - Application Pool: ApiPool
    - Enabled Protocols: http
    - Virtual Directories:
      - /:
        - Physical Path: C:\websites\api
        - Logon Method: ClearText
2: Intranet:
...
3. API.Internal:
...
4. Reports:
...
```

Understanding the discovery output

The verbose output provides the following critical information for migration planning:

Sites

The discovery output lists all IIS sites on your server. Each site is identified by its name (e.g., "Default Web Site", "Intranet", "API.Internal") and numbered sequentially. When multiple sites exist on a server, the `eb migrate` command can package and deploy each one separately or together, depending on your migration strategy.

Bindings

Protocol bindings reveal which protocols (HTTP/HTTPS) your sites use and on which ports they operate. The binding information includes host header requirements that define domain-based routing configurations.

Applications

Application paths show both root and nested application structures within your IIS configuration. Application pool assignments indicate how your applications are isolated from each other for security and resource management.

Virtual directories

Physical path mappings indicate where your content resides on the file system. Authentication settings show special access requirements that need to be maintained after migration.

Preparing for migration

With an understanding of your environment, ensure that your server meets the prerequisites. First, verify your IIS version with the following PowerShell command:

```
PS C:\migrations_workspace> Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\InetStp\" -Name
 MajorVersion
```

You need IIS 7.0 or later. The migration tool uses Web Deploy 3.6 for packaging your applications. Verify its installation with the following command:

```
PS C:\migrations_workspace> Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\IIS Extensions
\MSDeploy\3" -Name InstallPath
```

If Web Deploy isn't installed on your server, you can download it from the Microsoft Web Platform Installer download page.

Your first migration

Let's start with a basic migration of the Default Web Site. The following example shows the simplest command, **eb migrate**.

```
PS C:\migrations_workspace> eb migrate
```

This command initiates a series of automated steps, shown in the following example output:

```
Identifying VPC configuration of this EC2 instance (i-0123456789abcdef0)
  id: vpc-1234567890abcdef0
  publicip: true
  elbscheme: public
  ec2subnets: subnet-123,subnet-456,subnet-789
  securitygroups: sg-123,sg-456
  elbsubnets: subnet-123,subnet-456,subnet-789

Using .\migrations\latest to contain artifacts for this migration run.
Generating source bundle for sites, applications, and virtual directories...
  Default Web Site/ -> .\migrations\latest\upload_target\DefaultWebSite.zip
```

The migration tool creates a structured directory containing your deployment artifacts. The following listing shows the directory structure:

```
C:\migration_workspace\
### .\migrations\latest\
    ### upload_target\
        ### DefaultWebSite.zip
        ### aws-windows-deployment-manifest.json
        ### ebmigrateScripts\
            ### site_installer.ps1
            ### permission_handler.ps1
            ### >other helper scripts<
```

Controlling the migration

For more control over the migration process, you can specify exactly which sites to migrate with the following command:

```
PS C:\migrations_workspace> eb migrate --sites "Default Web Site,Intranet"
```

You can also customize the environment name and application name, as shown in the following example command:

```
PS C:\migrations_workspace> eb migrate `
```

```
    --sites "Default Web Site" `
    --application-name "CorporateApp" `
    --environment-name "Production"
```

For a complete list of options, see the section called “**eb migrate**”.

Monitoring progress

During migration, **eb migrate** provides real-time status updates. Refer to the following output example:

```
...
Creating application version
Creating environment... This may take a few minutes

2024-03-18 18:12:15    INFO    Environment details for: Production
  Application name: CorporateApp
  Region: us-west-2
  Deployed Version: app-230320_153045
  Environment ID: e-abcdef1234
  Platform: 64bit Windows Server 2019 v2.7.0 running IIS 10.0
  Tier: WebServer-Standard-1.0
  CNAME: production.us-west-2.elasticbeanstalk.com
  Updated: 2024-03-20 15:30:45
2025-03-18 18:12:17    INFO    createEnvironment is starting.
2025-03-18 18:12:19    INFO    Using elasticbeanstalk-us-east-1-180301529717 as Amazon
 S3 storage bucket for environment data.
2025-03-18 18:12:40    INFO    Created security group named: sg-0fdd4d696a26b086a
2025-03-18 18:12:48    INFO    Environment health has transitioned to Pending.
 Initialization in progress (running for 7 seconds). There are no instances.
...
2025-03-18 18:23:59    INFO    Application available at EBMigratedEnv-arrreal3.us-
east-1.elasticbeanstalk.com.
2025-03-18 18:24:00    INFO    Successfully launched environment: EBMigratedEnv-
arrreal3
```

Verifying the migration

Once the environment is ready, Elastic Beanstalk provides several ways to verify your deployment.

Access your application

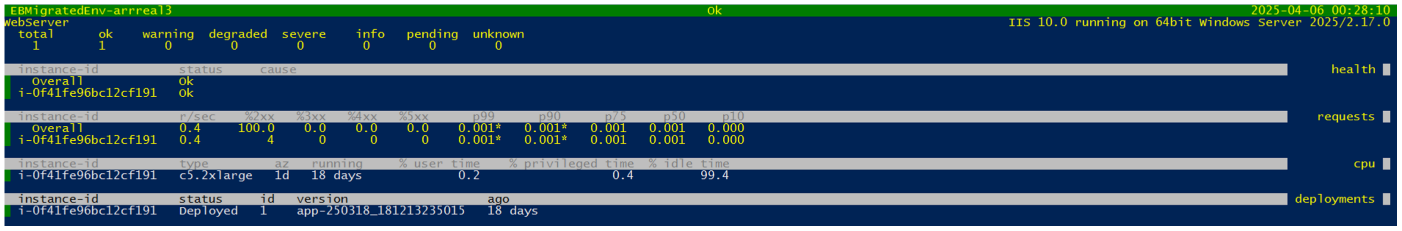
    Open your application URL (CNAME) in a web browser to verify it's working correctly.

Check environment health

Use the **eb health** command to view the health of your environment.

```
PS C:\migrations_workspace> eb health
```

The following screen image shows instance health, application response metrics, and system resource utilization.



```
 EBMigratedEnv-arrreal3                                      Ok                                   2025-04-06 00:28:10
WebServer                                                                 IIS 10.0 running on 64bit Windows Server 2025/2.17.0
  total      ok    warning  degraded  severe    info   pending  unknown
    1        1        0        0        0        0        0        0

  instance-id           status     cause                                                                       health
    Overall             Ok
  i-0f41fe96bc12cf191   Ok

  instance-id           r/sec    %2xx    %3xx    %4xx    %5xx      p99       p90      p75     p50      p10     requests
    Overall             0.4     100.0     0.0     0.0     0.0    0.001*    0.001*   0.001   0.001   0.000
  i-0f41fe96bc12cf191   0.4         4       0       0       0    0.001*    0.001*   0.001   0.001   0.000

  instance-id           type          az   running     % user time     % privileged time  % idle time         cpu
  i-0f41fe96bc12cf191   c5.2xlarge    1d   18 days            0.2                  0.4          99.4

  instance-id           status     id   version                    ago                                   deployments
  i-0f41fe96bc12cf191   Deployed   1    app-250318_181213235015   18 days
```

Use the **eb logs** command to access logs to troubleshoot any issues:

```
PS C:\migrations_workspace> eb logs --zip
```

The **eb logs** command downloads logs to the `.elasticbeanstalk/logs` directory. For more information, see the section called "CloudWatch Logs".

Connect to instances

If you specified a key pair during migration, you can connect to your instances using RDP for direct troubleshooting.

Access the Elastic Beanstalk console

You can view the environment's health, logs, and configuration properties through the environment management console for that environment.

Managing migration artifacts

The **eb migrate** command creates local artifacts during the migration process. These artifacts contain sensitive information and can consume significant disk space over time. Use the **cleanup** subcommand to manage these artifacts, as shown in the following example:

```
PS C:\migrations_workspace> eb migrate cleanup
Are you sure you would like to cleanup older artifacts within ./migrations/? (Y/N):
```

To force cleanup without confirmation use the **--force** option:

```
PS C:\migrations_workspace> eb migrate cleanup --force
```

The cleanup process preserves the most recent successful migration in the `./migrations/latest` directory and removes older migration directories

Network configuration and port settings

This section covers network configuration options for IIS migrations, including VPC settings, port configurations, and multi-site deployments.

VPC configuration

The **eb migrate** command provides flexible VPC configuration options for your Elastic Beanstalk environment. The tool can either detect VPC settings from a source EC2 instance or accept custom VPC configurations through command-line parameters. Review Using Elastic Beanstalk with Amazon VPC to understand how to configure Elastic Beanstalk with VPC.

Automatic VPC detection

When **eb migrate** runs on an EC2 instance, it automatically discovers and uses the VPC configuration from the source environment's EC2 instances. The following example output illustrates the configuration information that it detects:

```
PS C:\migrations_workspace > eb migrate
Identifying VPC configuration of this EC2 instance (i-0123456789abcdef0):
  id: vpc-1234567890abcdef0
  publicip: true
  elbscheme: public
  ec2subnets: subnet-123,subnet-456,subnet-789
  securitygroups: sg-123,sg-456
  elbsubnets: subnet-123,subnet-456,subnet-789
...
```

The detected configuration includes:

- VPC identifier
- Public IP assignment settings
- Load balancer scheme (public/private)
- EC2 instance subnet assignments

- Security group associations
- Load balancer subnet assignments

On-premises or non-AWS cloud hosts

When **eb migrate** runs from an on-premises server or a non-AWS cloud host, the Elastic Beanstalk service will use the default VPC in your AWS account. The following listing shows an example command and output:

```
PS C:\migrations_worspace> eb migrate `
      -k windows-test-pem `
      --region us-east-1 `
      -a EBMigratedEnv `
      -e EBMigratedEnv-test2 `
      --copy-firewall-config
Determining EB platform based on host machine properties
Using .\migrations\latest to contain artifacts for this migration run.
...
```

Review Using Elastic Beanstalk with Amazon VPC to understand how Elastic Beanstalk configures the default VPC for your environment.

Custom VPC configuration

For any source environment (EC2, on-premises, or non-AWS cloud) where you need specific VPC settings, provide a VPC configuration file like the one in the following example:

```
{
    "id": "vpc-12345678",
    "publicip": "true",
    "elbscheme": "public",
    "ec2subnets": ["subnet-a1b2c3d4", "subnet-e5f6g7h8"],
    "securitygroups": "sg-123456,sg-789012",
    "elbsubnets": ["subnet-a1b2c3d4", "subnet-e5f6g7h8"]
}
```

Apply this configuration using the following command:

```
PS C:\migrations_workspace> eb migrate --vpc-config vpc-config.json
```

Note

The VPC configuration file requires the `id` field that specifies the VPC ID. All other fields are optional, and Elastic Beanstalk will use default values for any fields that you don't specify.

Important

The migration will ignore any existing VPC settings from the source environment when you specify the `--vpc-config` *parameter*. When you use this parameter, the migration will only use the VPC settings specified in the configuration file that you're passing in. Using this parameter overrides the default behavior of discovering the source instance's VPC configuration or using the default VPC.

Use the `--vpc-config` parameter in these scenarios:

- When you migrate non-EC2 environments that don't have discoverable VPC settings
- When you migrate to a different VPC from the one used by the source environment
- When you need to customize subnet selections or security group configurations
- When the automatic discovery doesn't correctly identify the desired VPC settings
- When you migrate from on-premises, and you don't want to use the default VPC

Network security configuration

By default, **eb migrate** opens port 80 on target instances but does not copy other Windows Firewall rules from the source machine. To include all firewall configurations use the following command:

```
PS C:\migrations_workspace> eb migrate --copy-firewall-config
```

This command does the following actions:

- Identifies ports used by IIS site bindings
- Retrieves corresponding firewall rules
- Generates PowerShell scripts to recreate rules on target instances

- Preserves any DENY rules for port 80 from the source machine (otherwise port 80 is allowed by default)

Consider a use case, where your source machine has the firewall rules specified in the following example:

```
# Source machine firewall configuration
Get-NetFirewallRule | Where-Object {$_.Enabled -eq 'True'} | Get-NetFirewallPortFilter
 | Where-Object {$_.LocalPort -eq 80 -or $_.LocalPort -eq 443 -or $_.LocalPort -eq
 8081}
# Output shows rules for ports 80, 443, and 8081
```

The migration creates a script (`modify_firewall_config.ps1`) that contains the following configuration:

```
New-NetFirewallRule -DisplayName "Allow Web Traffic" -Direction Inbound -Action Allow -
Protocol TCP -LocalPort 80,443
New-NetFirewallRule -DisplayName "Allow API Traffic" -Direction Inbound -Action Allow -
Protocol TCP -LocalPort 8081
```

The migration tool automatically does the following actions:

- Extracts HTTP/HTTPS ports from all IIS site bindings
- Uses the Windows Firewall INetFwPolicy2 interface to enumerate firewall rules
- Filters rules to only include those that explicitly reference the specified ports
- Only processes HTTP and HTTPS site bindings and their associated firewall rules
- Preserves rule properties including display name, action, protocol, and enabled state
- Handles both individual ports and port ranges in firewall rules
- Adds the firewall configuration script to the deployment manifest

Load balancer configuration

You can specify Load Balancer configuration through the `--vpc-config` argument. The examples that follow demonstrate the parameters.

Scheme selection

Choose between public and private load balancer schemes:

```
{
    "id": "vpc-12345678",
    "elbscheme": "private",
    "elbsubnets": ["subnet-private1", "subnet-private2"]
}
```

Subnet distribution

For high availability, distribute load balancer subnets across Availability Zones:

```
{
    "elbsubnets": [
        "subnet-az1", // Availability Zone 1
        "subnet-az2", // Availability Zone 2
        "subnet-az3"  // Availability Zone 3
    ]
}
```

Note

While Elastic Beanstalk supports environment creation with Application Load Balancers, Network Load Balancers, and Classic Load Balancers, the **eb migrate** command only supports Application Load Balancers. For more information about load balancer types, see Load balancer for your Elastic Beanstalk environment.

Multi-site deployments with port configurations

The **eb migrate** command handles complex multi-site IIS deployments where applications might share dependencies or use non-standard ports. Consider the following example of a typical enterprise setup with multiple sites:

```
<!-- IIS Configuration -->
<sites>
    <site name="Default Web Site" id="1">
        <bindings>
            <binding protocol="http" bindingInformation="*:80:www.example.com" />
        </bindings>
    </site>
    <site name="InternalAPI" id="2">
```

```
        <bindings>
            <binding protocol="http" bindingInformation="*:8081:api.internal" />
        </bindings>
    </site>
    <site name="ReportingPortal" id="3">
        <bindings>
            <binding protocol="http" bindingInformation="*:8082:reports.internal" />
        </bindings>
    </site>
</sites>
```

To migrate this configuration, use the following example command and parameters:

```
PS C:\migrations_workspace> eb migrate `
    --sites "Default Web Site,InternalAPI,ReportingPortal" `
    --copy-firewall-config `
    --instance-type "c5.large"
```

The **eb migrate** command creates a deployment package that preserves each site's identity and configuration. The command generates an `aws-windows-deployment-manifest.json` that defines how these sites should be deployed. The following example demonstrates a generated json file:

```
{
    "manifestVersion": 1,
    "deployments": {
        "msDeploy": [
            {
                "name": "DefaultWebSite",
                "parameters": {
                    "appBundle": "DefaultWebSite.zip",
                    "iisPath": "/",
                    "iisWebSite": "Default Web Site"
                }
            }
        ],
        "custom": [
            {
                "name": "InternalAPI",
                "scripts": {
                    "install": {
                        "file": "ebmigrateScripts\\install_site_InternalAPI.ps1"
                    },
                    "restart": {
                        "file": "ebmigrateScripts\\restart_site_InternalAPI.ps1"
                    },
                    "uninstall": {
                        "file": "ebmigrateScripts\\uninstall_site_InternalAPI.ps1"
                    }
                }
            },
            {
                "name": "ReportingPortal",
                "scripts": {
                    "install": {
                        "file": "ebmigrateScripts\\install_site_ReportingPortal.ps1"
                    },
                    "restart": {
                        "file": "ebmigrateScripts\\restart_site_ReportingPortal.ps1"
                    },
                    "uninstall": {
                        "file": "ebmigrateScripts\\uninstall_site_ReportingPortal.ps1"
                    }
                }
            }
        ]
    }
}
```

The migration process creates the following Application Load Balancer listener rules that maintain your original routing logic:

- Port 80 traffic routes to Default Web Site
- Port 8081 traffic routes to InternalAPI
- Port 8082 traffic routes to ReportingPortal

Shared configuration and dependencies

When sites share configurations or dependencies, **eb migrate** handles these relationships appropriately. Reference the following example where multiple sites share a common configuration:

```
<!-- Shared configuration in applicationHost.config -->
```

```
<location path="Default Web Site">
    <system.webServer>
        <asp enableSessionState="true" />
        <caching enabled="true" enableKernelCache="true" />
    </system.webServer>
</location>
```

The migration process completes the following tasks:

1. Identifies shared configurations across sites
2. Generates appropriate PowerShell scripts to apply these settings
3. Maintains configuration hierarchy and inheritance

Best practices

We recommend that you follow best practices for the network configuration of your migrated application. The following groupings provide summary guidelines.

VPC design

- Follow AWS VPC Design Best Practices
- Use separate subnets for load balancers and EC2 instances
- Implement proper route tables and NACLs
- Consider VPC endpoints for AWS services

High availability

- Deploy across multiple Availability Zones
- Use at least two subnets for load balancers
- Configure auto-scaling across AZs
- Implement proper health checks

Security

- Follow Security Best Practices
- Use security groups as primary access control
- Implement network Access Control Lists (ACLs) for additional security
- Monitor VPC Flow Logs

Troubleshooting

Common network configuration issues include the following areas. Following each subject are example commands to obtain more information about the network configuration and health of your environment.

Subnet configuration

```
# Verify subnet availability
PS C:\migrations_workspace> aws ec2 describe-subnets --subnet-ids subnet-id

# Check available IP addresses
PS C:\migrations_workspace>aws ec2 describe-subnets --subnet-ids subnet-id --query
 'Subnets[].AvailableIpAddressCount'
```

Security group access

```
# Verify security group rules
PS C:\migrations_workspace> aws ec2 describe-security-groups --group-ids sg-id

# Test network connectivity
PS C:\migrations_workspace> aws ec2 describe-network-interfaces --filters
 Name=group-id,Values=sg-id
```

Load balancer health

```
# Check load balancer health
PS C:\migrations_workspace> aws elbv2 describe-target-health --target-group-arn
 arn:aws:elasticloadbalancing:region:account-id:targetgroup/group-name/group-id
```

Security configurations and IAM roles

The **eb migrate** command manages AWS security configurations through IAM roles, instance profiles, and service roles. Understanding these components ensures proper access control and security compliance during migration.

Instance profile configuration

An instance profile serves as a container for an IAM role that Elastic Beanstalk attaches to EC2 instances in your environment. When executing **eb migrate**, you can specify a custom instance profile:

```
PS C:\migrations_workspace> eb migrate --instance-profile "CustomInstanceProfile"
```

If you don't specify an instance profile, **eb migrate** creates a default profile with these permissions:

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::elasticbeanstalk-*",
                "arn:aws:s3:::elasticbeanstalk-*/*"
            ]
        }
    ]
}
```

Service role management

A service role allows Elastic Beanstalk to manage AWS resources on your behalf. Specify a custom service role during migration with the following command:

```
PS C:\migrations_workspace> eb migrate --service-role "CustomServiceRole"
```

If not specified, **eb migrate** creates a default service role named `aws-elasticbeanstalk-service-role` with a trust policy that allows Elastic Beanstalk to assume the role. This service role is essential for Elastic Beanstalk to monitor your environment's health and perform managed platform updates. The service role requires two managed policies:

- `AWSElasticBeanstalkEnhancedHealth` - Allows Elastic Beanstalk to monitor instance and environment health using the enhanced health reporting system
- `AWSElasticBeanstalkManagedUpdates` - Allows Elastic Beanstalk to perform managed platform updates, including updating environment resources when a new platform version is available

With these policies, the service role has permissions to:

- Create and manage Auto Scaling groups
- Create and manage Application Load Balancers
- Upload logs to Amazon CloudWatch
- Manage EC2 instances

For more information about service roles, see Elastic Beanstalk service role in the Elastic Beanstalk Developer Guide.

Security group configuration

The **eb migrate** command automatically configures security groups based on your IIS site bindings. For example, if your source environment has sites using ports 80, 443, and 8081 the following configuration results:

```
<site name="Default Web Site">
    <bindings>
        <binding protocol="http" bindingInformation="*:80:" />
        <binding protocol="https" bindingInformation="*:443:" />
    </bindings>
</site>
<site name="InternalAPI">
    <bindings>
        <binding protocol="http" bindingInformation="*:8081:" />
    </bindings>
</site>
```

The migration process completes the following actions:

- Creates a load balancer security group allowing inbound traffic on ports 80 and 443 from the internet (0.0.0.0/0)
- Creates an EC2 security group allowing traffic from the load balancer
- Configures additional ports (like 8081) if `--copy-firewall-config` is specified

By default, the Application Load Balancer is configured with public access from the internet. If you need to customize this behavior, such as restricting access to specific IP ranges or using a private load balancer, you can override the default VPC and security group configuration using the `--vpc-config` parameter:

```
PS C:\migrations_workspace> eb migrate --vpc-config vpc-config.json
```

For example, the following `vpc-config.json` configuration creates a private load balancer in a private subnet:

```
{
    "id": "vpc-12345678",
    "publicip": "false",
    "elbscheme": "internal",
    "ec2subnets": ["subnet-private1", "subnet-private2"],
    "elbsubnets": ["subnet-private1", "subnet-private2"]
}
```

For more information about VPC configuration options, see VPC configuration.

SSL certificate integration

When migrating sites with HTTPS bindings, integrate SSL certificates through AWS Certificate Manager (ACM):

```
PS C:\migrations_workspace> eb migrate --ssl-certificates
 "arn:aws:acm:region:account:certificate/certificate-id"
```

This configuration completes the following actions:

- Associates the certificate with the Application Load Balancer
- Maintains HTTPS termination at the load balancer
- Preserves internal HTTP communication between the load balancer and EC2 instances

Windows authentication

For applications using Windows Authentication, **eb migrate** preserves the authentication settings in the application's `web.config` as follows:

```
<configuration>
    <system.webServer>
        <security>
            <authentication>
                <windowsAuthentication enabled="true">
                    <providers>
                        <add value="Negotiate" />
                        <add value="NTLM" />
                    </providers>
                </windowsAuthentication>
            </authentication>
        </security>
    </system.webServer>
</configuration>
```

Important

The **eb migrate** command does not copy over user profiles or accounts from your source environment to the target Elastic Beanstalk instances. Any custom user accounts or groups that you've created on your source server will need to be recreated on the target environment after migration.

Built-in Windows accounts like `IUSR` and groups like `IIS_IUSRS`, as well as all other built-in accounts and groups, are included by default on the target Windows Server instances. For more information about built-in IIS accounts and groups, see Understanding Built-In User and Group Accounts in IIS in the Microsoft documentation.

If your application relies on custom Windows user accounts or Active Directory integration, you will need to configure these aspects separately after the migration is complete.

Best practices and troubleshooting

Role management

Implement AWS IAM best practices when managing roles for your Elastic Beanstalk environments:

Role creation and management

- Create roles using AWS managed policies where possible
- Follow the IAM Security Best Practices
- Use the AWS Policy Generator for custom policies
- Implement permission boundaries for additional security

Monitoring and auditing

Enable AWS CloudTrail to monitor role usage:

- Follow the AWS CloudTrail User Guide
- Configure CloudWatch Logs integration for real-time monitoring
- Set up alerts for unauthorized API calls

Regular review process

Establish a quarterly review cycle to do the following tasks:

- Audit unused permissions using IAM Access Analyzer
- Remove outdated permissions
- Update roles based on least-privilege principles

Certificate management

Implement these practices for SSL/TLS certificates in your Elastic Beanstalk environments:

Certificate lifecycle

- Use AWS Certificate Manager for certificate management
- Enable automatic renewal for ACM-issued certificates
- Set up expiration notifications

Security standards

- Use TLS 1.2 or later
- Follow AWS security policies for HTTPS listeners
- Implement HTTP Strict Transport Security (HSTS) if required

Security group management

Implement these security group best practices:

Rule management

- Document all custom port requirements
- Use VPC Flow Logs to monitor traffic
- Use Security Group reference rules instead of IP ranges where possible

Regular auditing

Establish monthly reviews to do the following tasks:

- Identify and remove unused rules
- Validate source/destination requirements
- Check for overlapping rules

Logging and monitoring

For effective security monitoring, configure the following logs:

Windows event logs on EC2 instances

```
# Review Security event log
PS C:\migrations_workspace> Get-EventLog -LogName Security -Newest 50

# Check Application event log
PS C:\migrations_workspace> Get-EventLog -LogName Application -Source "IIS*"
```

CloudWatch Logs integration

Configure CloudWatch Logs agent to stream Windows event logs to CloudWatch for centralized monitoring and alerting.

For persistent issues, gather these logs and contact AWS Support with the following information:

- Environment ID
- Deployment ID (if applicable)
- Relevant error messages
- Timeline of security changes

Understanding IIS to Elastic Beanstalk migration mapping

The migration from IIS to Elastic Beanstalk involves mapping your on-premises Windows server configuration to AWS cloud resources. Understanding this mapping is crucial for successful migrations and post-migration management.

IIS sites and applications in Elastic Beanstalk

In IIS, a website represents a collection of web applications and virtual directories, each with its own configuration and content. When migrating to Elastic Beanstalk, these components are transformed as follows:

IIS websites

Your IIS websites become applications within Elastic Beanstalk. Each website's configuration, including its bindings, application pools, and authentication settings, is preserved through Elastic Beanstalk's deployment manifest (`aws-windows-deployment-manifest.json`).

For example, if you have multiple sites like *Default Web Site* and *IntranetSite*, **eb migrate** packages each site's content and configuration while maintaining their isolation.

The command creates appropriate Application Load Balancer (ALB) listener rules to handle routing requests to your applications. It also configures security groups to ensure proper port access based on your original IIS bindings.

Application pools

IIS application pools provide worker process isolation, runtime management, and recycling capabilities for your applications. In Elastic Beanstalk, these are mapped to environment processes defined through the `aws:elasticbeanstalk:environment:process` namespace and configured via IIS on the EC2 instances.

The migration preserves critical application pool settings including the following:

- Process model configurations - Identity (ApplicationPoolIdentity, NetworkService, or custom accounts), idle timeout settings, and process recycling intervals
- .NET CLR version settings - Maintains your specified .NET Framework version (v2.0, v4.0, or No Managed Code) to ensure application compatibility
- Managed pipeline mode - Preserves Integrated or Classic pipeline mode settings to maintain your HTTP request processing architecture

- Advanced settings - Queue length, CPU limits, rapid-fail protection thresholds, and startup time limits

The **eb migrate** command preserves mappings between sites and application pools during the migration to your Elastic Beanstalk environment.

If your application pools use custom recycling schedules (specific times or memory thresholds), these are implemented through PowerShell scripts in the deployment package that configure the appropriate IIS settings on the EC2 instances.

Website bindings

IIS website bindings, which define how clients access your applications, are transformed into the following Application Load Balancer (ALB) configurations:

- Port bindings are mapped to corresponding ALB listener rules
- Host header configurations are translated into ALB routing rules
- SSL-enabled sites use AWS Certificate Manager (ACM) for certificate management

Virtual directory and application path management

IIS virtual directories and applications provide URL path mapping to physical directories. Elastic Beanstalk maintains these relationships through the following constructs:

Virtual directories

The migration process preserves the physical paths of your virtual directories in the deployment package.

Path mappings are configured in the IIS configuration on the EC2 instances, ensuring that your URL structure remains intact after migration.

Non-system drive physical paths

Important

By default, Elastic Beanstalk Windows environments only provision the C:\ drive (root volume). In the current version, applications with content on non-system drives (D:\, E:\, etc.) are not supported for migration.

The **eb migrate** command automatically detects physical paths located on non-system drives and warns you about potential issue like the following example:

```
ERROR: Detected physical paths on drive D:\ which are not supported in the current
 version:
  - D:\websites\intranet
  - D:\shared\images

Migration of content from non-system drives is not supported. Please relocate this
 content to the C:\ drive before migration. Otherwise, select only those sites that
 are on C:\.
```

If your application has dependencies on non-system drives, you will need to modify your application to store all content on the C:\ drive before migration.

Nested applications

Applications nested under websites are deployed with their correct path configurations and appropriate application pool assignments. The migration process preserves all `web.config` settings, ensuring that application-specific configurations continue to function as expected in the cloud environment.

URL rewrite and application request routing (ARR)

If your IIS deployment uses URL Rewrite or Application Request Routing (ARR), **eb migrate** handles these configurations through the following rules and configuration:

URL rewrite rules

URL rewrite rules from your `web.config` files are translated into ALB routing rules where possible. For example, the following entry becomes an ALB listener rule directing traffic based on host headers and path patterns.:

```
<!-- Original IIS URL Rewrite Rule -->
<rule name="Redirect to WWW" stopProcessing="true">
    <match url="(.*)" />
    <conditions>
        <add input="{HTTP_HOST}" pattern="^example.com$" />
    </conditions>
    <action type="Redirect" url="http://www.example.com/{R:1}" />
</rule>
```

Application request routing

ARR configurations are preserved through the installation of ARR features on EC2 instances. The migration process completes the following tasks:

- Configures proxy settings to match your source environment
- Maintains URL rewrite rules associated with ARR

Migration artifact structure

When you run **eb migrate**, it creates a structured directory containing all necessary deployment components. The following listing describes the directory structure:

```
C:\migration_workspace\
### .\migrations\latest\
    ### upload_target\
        ### [SiteName].zip                 # One ZIP per IIS site
        ### aws-windows-deployment-manifest.json
        ### ebmigrateScripts\
            ### site_installer.ps1         # Site installation scripts
            ### arr_configuration.ps1      # ARR configuration scripts
            ### permission_handler.ps1     # Permission management
            ### firewall_config.ps1        # Windows Firewall rules
```

The `aws-windows-deployment-manifest.json` file is the core configuration file that instructs Elastic Beanstalk how to deploy your applications. Refer to the following example structure:

```
{
    "manifestVersion": 1,
    "deployments": {
        "msDeploy": [
            {
                "name": "Primary Site",
                "parameters": {
                    "appBundle": "DefaultWebSite.zip",
                    "iisPath": "/",
                    "iisWebSite": "Default Web Site"
                }
            }
        ],
        "custom": [
            {
                "name": "ConfigureARR",
                "scripts": {
                    "install": {
                        "file": "ebmigrateScripts\\arr_configuration.ps1"
                    },
                    "uninstall": {
                        "file": "ebmigrateScripts\\noop.ps1"
                    },
                    "restart": {
                        "file": "ebmigrateScripts\\noop.ps1"
                    }
                }
            }
        ]
    }
}
```

This manifest ensures these results for your migration:

- Applications are deployed to correct IIS paths
- Custom configurations are applied
- Site-specific settings are preserved
- Deployment order is maintained

Advanced migration scenarios

This section covers advanced migration scenarios for complex IIS deployments.

Multi-site migrations with Application Request Routing (ARR)

The **eb migrate** command automatically detects and preserves ARR configurations during migration. When it identifies ARR settings in your IIS `applicationHost.config`, it generates the necessary PowerShell scripts to reinstall and configure ARR on the target EC2 instances.

ARR configuration detection

The migration process examines three key configuration sections in IIS:

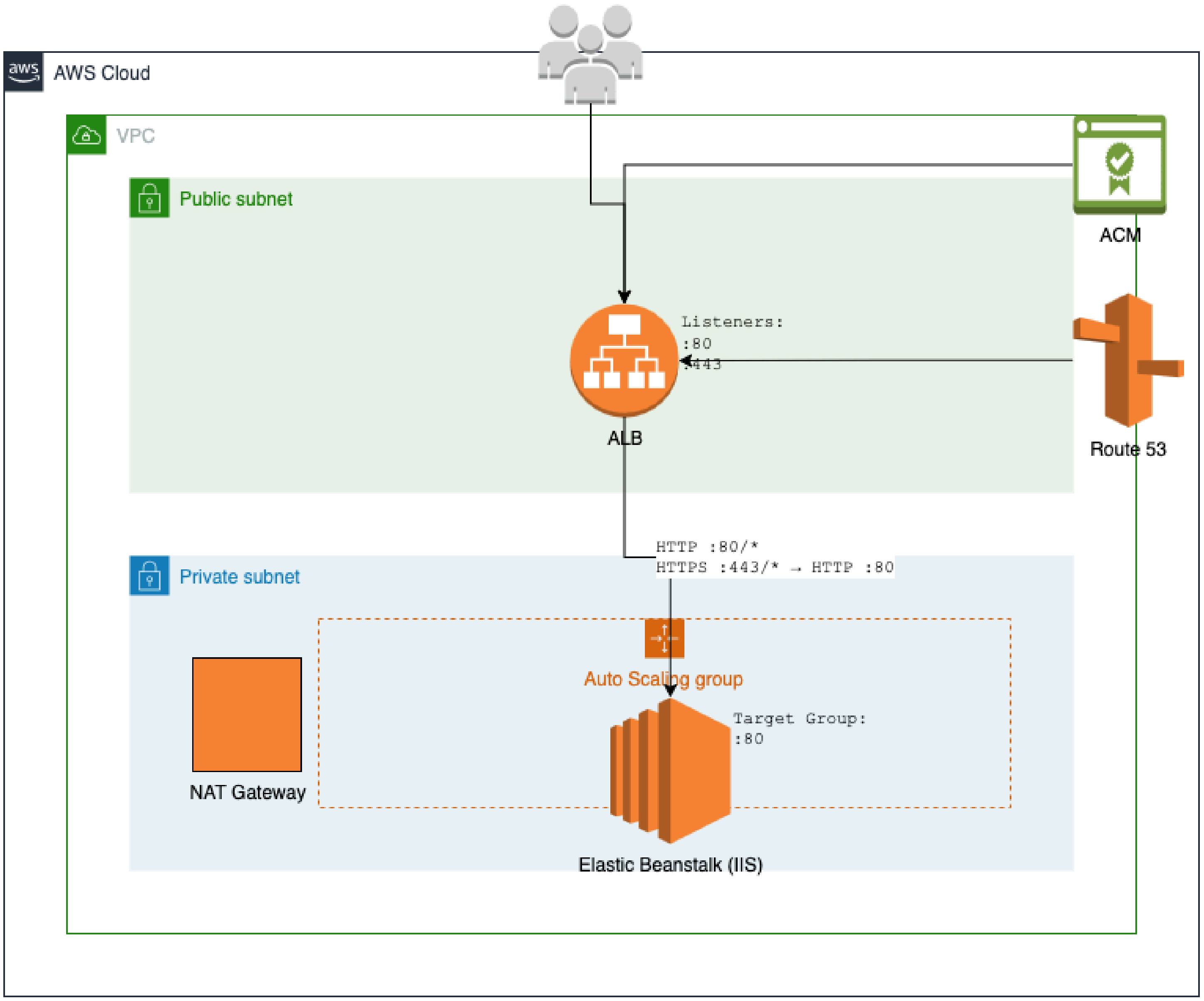
- `system.webServer/proxy`: Core ARR proxy settings

- `system.webServer/rewrite`: URL rewrite rules
- `system.webServer/caching`: Caching configuration

For example, consider a common ARR configuration where a `RouterSite` running on port 80 proxies requests to `APIService` and `AdminPortal` running on ports 8081 and 8082 respectively:

```
<!-- Original IIS ARR Configuration -->
<rewrite>
    <rules>
        <rule name="Route to API" stopProcessing="true">
            <match url="^api/(.*)$" />
            <action type="Rewrite" url="http://backend:8081/api/{R:1}" />
        </rule>
        <rule name="Route to Admin" stopProcessing="true">
            <match url="^admin/(.*)$" />
            <action type="Rewrite" url="http://backend:8082/admin/{R:1}" />
        </rule>
    </rules>
</rewrite>
```

The following diagram depicts how these rules are hidden behind port 80 in the IIS server and not exposed via the EC2 Security Groups. Only port 80 is accessible to the Application Load Balancer and all traffic from it is routed to the target group at port 80.



The following command can migrate this configuration:

```
PS C:\migrations_workspace> eb migrate --sites "RouterSite,APIService,AdminPortal" `
    --copy-firewall-config
```

ARR migration process

The migration process preserves your ARR configuration through several steps.

Configuration export

The tool exports your existing ARR settings from the three key configuration sections into separate XML files stored in the `ebmigrateScripts` directory:

```
ebmigrateScripts\
### arr_config_proxy.xml
### arr_config_rewrite.xml
### arr_config_caching.xml
```

Installation scripts

Two PowerShell scripts are generated to handle ARR setup:

1. `arr_msi_installer.ps1`: Downloads and installs the ARR module
2. `arr_configuration_importer_script.ps1`: Imports your exported ARR configuration

Deployment manifest integration

The scripts are integrated into the deployment process through entries in `aws-windows-deployment-manifest.json`:

```
{
    "manifestVersion": 1,
    "deployments": {
        "custom": [
            {
                "name": "WindowsProxyFeatureEnabler",
                "scripts": {
                    "install": {
                        "file": "ebmigrateScripts\
\windows_proxy_feature_enabler.ps1"
                    }
                }
            },
            {
                "name": "ArrConfigurationImporterScript",
                "scripts": {
                    "install": {
                        "file": "ebmigrateScripts\
\arr_configuration_importer_script.ps1"
                    }
                }
            }
        ]
    }
}
```

Load balancer integration

The migration process translates your ARR rules into Application Load Balancer (ALB) listener rules where possible. For example, the above ARR configuration results in ALB rules that route traffic based on URL path patterns while maintaining internal routing on the EC2 instances.

The resulting environment maintains your ARR routing logic while taking advantage of AWS's elastic infrastructure. Your applications continue to work as before, with ARR handling internal routing while the Application Load Balancer manages external traffic distribution.

Multi-site migrations without ARR using host-based routing

While Application Request Routing (ARR) is a common approach for managing multiple sites in IIS, you can also migrate multi-site deployments directly to Elastic Beanstalk without ARR by leveraging the Application Load Balancer's host-based routing capabilities. This approach can reduce complexity and improve performance by eliminating an additional routing layer.

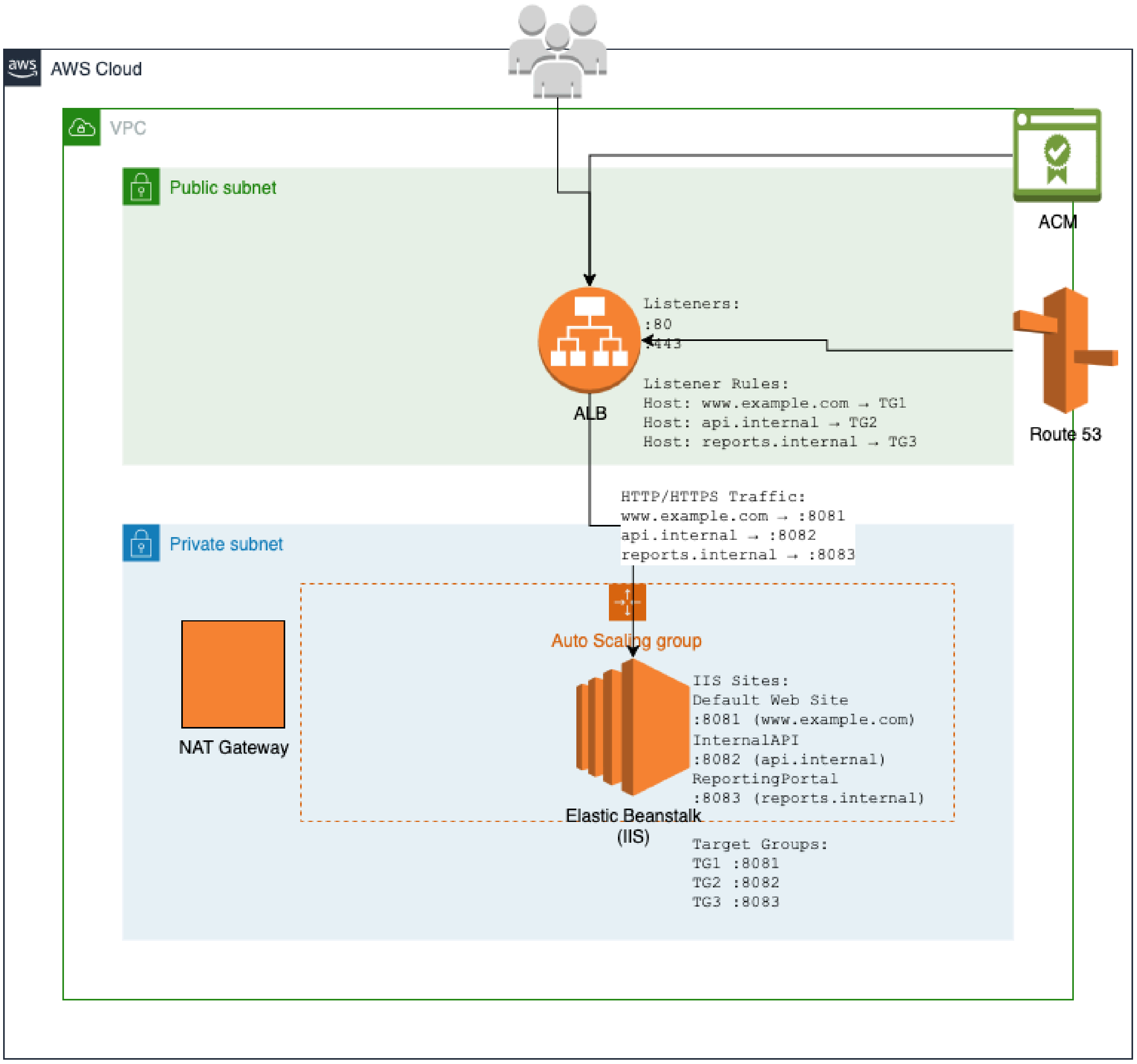
Host-based routing overview

In this approach, each IIS site is exposed outside the EC2 instance, and the Application Load Balancer routes traffic directly to the appropriate port based on the host header. This eliminates the need for ARR while maintaining separation between your applications.

Consider a multi-site IIS configuration with three sites, each with its own hostname binding:

```
<sites>
    <site name="Default Web Site" id="1">
        <bindings>
            <binding protocol="http" bindingInformation="*:8081:www.example.com" />
        </bindings>
    </site>
    <site name="InternalAPI" id="2">
        <bindings>
            <binding protocol="http" bindingInformation="*:8082:api.internal" />
        </bindings>
    </site>
    <site name="ReportingPortal" id="3">
        <bindings>
            <binding protocol="http" bindingInformation="*:8083:reports.internal" />
        </bindings>
    </site>
</sites>
```

These sites are exposed at ports 8081, 8082, and 8083 via the EC2 Security Groups. The Application Load Balancer routes to them based on the Load Balancer listener rule configuration.



Migration process

To migrate this configuration to Elastic Beanstalk without using ARR use the **eb migrate** command in the following example:

```
PS C:\migrations_workspace> eb migrate --sites "Default Web
 Site,InternalAPI,ReportingPortal"
```

The migration process automatically configures the Application Load Balancer with host-based routing rules that direct traffic to the appropriate target group based on the host header. Each target group forwards traffic to the corresponding port on your EC2 instances:

1. Host header www.example.com → Target Group on port 8081
2. Host header api.internal → Target Group on port 8082
3. Host header reports.internal → Target Group on port 8083

SSL/TLS configuration

To secure your applications with SSL/TLS do the following steps:

1. Request certificates for your domains through AWS Certificate Manager(ACM).
2. Configure HTTPS listeners on your Application Load Balancer using these certificates.
3. Update your environment configuration to include HTTPS listeners with the following configuration option settings.

```
option_settings:
  aws:elb:listener:443:
    ListenerProtocol: HTTPS
    SSLCertificateId: arn:aws:acm:region:account-id:certificate/certificate-id
    InstancePort: 80
    InstanceProtocol: HTTP
```

With this configuration, SSL termination occurs at the load balancer, and traffic is forwarded to your instances over HTTP. This simplifies certificate management while maintaining secure connections with clients.

Best practices

Security groups

Configure security groups to allow inbound traffic only on the ports used by your IIS sites (8081, 8082, 8083 in this example) from the Application Load Balancer security group.

Health checks

Configure health checks for each target group to ensure traffic is only routed to healthy instances. Create health check endpoints for each application if they don't already exist.

Monitoring

Set up CloudWatch alarms to monitor the health and performance of each target group separately. This allows you to identify issues specific to individual applications.

Scaling

Consider the resource requirements of all applications when configuring auto scaling policies. If one application has significantly different resource needs, consider migrating it to a separate environment.

Virtual directory management

The **eb migrate** command preserves virtual directory structures while migrating your IIS applications to Elastic Beanstalk.

Default permission configuration

When migrating virtual directories, **eb migrate** establishes a baseline set of permissions by granting ReadAndExecute access to:

- IIS_IUSRS
- IUSR
- Authenticated Users

For example, consider a typical virtual directory structure:

```
<site name="CorporatePortal">
    <application path="/" applicationPool="CorporatePortalPool">
```

```
        <virtualDirectory path="/" physicalPath="C:\sites\portal" />
        <virtualDirectory path="/shared" physicalPath="C:\shared\content" />
        <virtualDirectory path="/reports" physicalPath="D:\reports" />
    </application>
</site>
```

Password-protected virtual directories

When **eb migrate** encounters password-protected virtual directories, it issues warnings and requires manual intervention.

The following configuration example will cause the warning response that follows the example.

```
<virtualDirectory path="/secure"
                 physicalPath="C:\secure\content"
                 userName="DOMAIN\User"
                 password="[encrypted]" />
```

```
[WARNING] CorporatePortal/secure is hosted at C:\secure\content which is password-
protected and won't be copied.
```

To maintain password protection, create a custom deployment script like the following:

```
# PS C:\migrations_workspace> cat secure_vdir_config.ps1

$vdirPath = "C:\secure\content"
$siteName = "CorporatePortal"
$vdirName = "secure"
$username = "DOMAIN\User"
$password = "SecurePassword"

# Ensure directory exists
if (-not (Test-Path $vdirPath)) {
    Write-Host "Creating directory: $vdirPath"
    New-Item -Path $vdirPath -ItemType Directory -Force
}

# Configure virtual directory with credentials
Write-Host "Configuring protected virtual directory: $vdirName"
New-WebVirtualDirectory -Site $siteName -Name $vdirName `
    -PhysicalPath $vdirPath -UserName $username -Password $password
```

```
# Set additional permissions as needed
$acl = Get-Acl $vdirPath
$rule = New-Object System.Security.AccessControl.FileSystemAccessRule(
    $username, "ReadAndExecute", "ContainerInherit,ObjectInherit", "None", "Allow"
)
$acl.AddAccessRule($rule)
Set-Acl $vdirPath $acl
```

Add this script to your deployment by including it in the manifest:

```
{
    "manifestVersion": 1,
    "deployments": {
        "custom": [
            {
                "name": "SecureVirtualDirectory",
                "scripts": {
                    "install": {
                        "file": "secure_vdir_config.ps1"
                    }
                }
            }
        ]
    }
}
```

Custom permission management

The **eb migrate** command provides a framework for custom permission scripts to accommodate applications that require permissions other than the defaults.

```
$paths = @(
    "C:\sites\portal\uploads",
    "C:\shared\content"
)

foreach ($path in $paths) {
    if (-not (Test-Path $path)) {
        Write-Host "Creating directory: $path"
        New-Item -Path $path -ItemType Directory -Force
    }
    $acl = Get-Acl $path

    # Add custom permissions
    $customRules = @(
        # Application Pool Identity - Full Control
        [System.Security.AccessControl.FileSystemAccessRule]::new(
            "IIS AppPool\CorporatePortalPool",
            "FullControl",
            "ContainerInherit,ObjectInherit",
            "None",
            "Allow"
        ),
        # Custom Service Account
        [System.Security.AccessControl.FileSystemAccessRule]::new(
            "NT SERVICE\CustomService",
            "Modify",
            "ContainerInherit,ObjectInherit",
            "None",
            "Allow"
        )
    )

    foreach ($rule in $customRules) {
        $acl.AddAccessRule($rule)
    }

    Set-Acl $path $acl
    Write-Host "Custom permissions applied to: $path"
}
```

Best practices

Follow these best practices to plan, execute, monitor, and verify your migration.

Pre-migration planning

Document existing permissions and authentication requirements before migration. Test custom permission scripts in a development environment before deploying to production.

Shared content management

For shared content directories, ensure all necessary file system permissions are properly configured through custom scripts. Consider using Amazon FSx for Windows File Server for shared storage requirements.

Monitoring and verification

Monitor application logs after migration to verify proper access to virtual directories. Pay special attention to the following areas:

- Application pool identity access
- Custom service account permissions
- Network share connectivity
- Authentication failures

Custom application pool settings

The **eb migrate** command does not copy over custom application pool settings by default. To preserve custom application pool configurations, follow this procedure to create and apply a custom manifest section.

1. Create an archive of your migration artifacts.

```
PS C:\migrations_workspace> eb migrate --archive
```

2. Create a custom PowerShell script to configure application pools.

```
# PS C:\migrations_workspace> cat .\migrations\latest\upload_target
\customize_application_pool_config.ps1

$configPath = "$env:windir\System32\inetsrv\config\applicationHost.config"

[xml]$config = Get-Content -Path $configPath

$newPoolXml = @"
<!-- Original IIS Configuration -->
<applicationPools>
    <add name="CustomPool"
         managedRuntimeVersion="v4.0"
         managedPipelineMode="Integrated">
        <processModel identityType="SpecificUser"
                     userName="AppPoolUser"
                     password="[encrypted]" />
        <recycling>
            <periodicRestart time="00:00:00">
                <schedule>
```

```
                    <add value="02:00:00" />
                    <add value="14:00:00" />
                </schedule>
            </periodicRestart>
        </recycling>
    </add>
</applicationPools>
"@
$newPoolXmlNode = [xml]$newPoolXml

# Find the applicationPools section
$applicationPools = $config.SelectSingleNode("//configuration/
system.applicationHost/applicationPools")

# Import the new node into the document
$importedNode = $config.ImportNode($newPoolXmlNode.DocumentElement, $true)
$applicationPools.AppendChild($importedNode)

# Save the changes
$config.Save($configPath)

Write-Host "ApplicationHost.config has been updated successfully."
```

3. Update the `aws-windows-deployment-manifest.json` file to include your custom script.

```
{
    "manifestVersion": 1,
    "deployments": {
        ...
        "custom": [
            ...,
            {
                "name": "ModifyApplicationPoolConfig",
                "description": "Modify application pool configuration from source
 machine to remove",
                "scripts": {
                    "install": {
                        "file": "customize_application_pool_config.ps1"
                    },
                    "restart": {
                        "file": "ebmigrateScripts\\noop.ps1"
                    },
                    "uninstall": {
                        "file": "ebmigrateScripts\\noop.ps1"
                    }
                }
            }
        ]
    }
}
```

4. Create an environment with the updated archive directory.

```
PS C:\migrations_workspace> eb migrate `
    --archive-dir '.\migrations\latest\upload_target\'
```

The `--archive-dir` argument tells **eb migrate** to use the source code that it previously created, avoiding the creation of new archives.

Deploying previous versions

The **eb migrate** maintains a history of your migrations through timestamped directories and application versions in Elastic Beanstalk. Each migration creates a unique zip file that can be deployed if needed.

```
PS C:\migrations_workspace> ls .\migrations\

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----l        3/18/2025  10:34 PM                latest
d-----        3/16/2025   5:47 AM                migration_1742104049.479849
d-----        3/17/2025   9:18 PM                migration_1742246303.18056
d-----        3/17/2025   9:22 PM                migration_1742246546.565739
...
d-----        3/18/2025  10:34 PM                migration_1742337258.30742
```

The `latest` symbolic link always points to the most recently created migration artifact directory. In addition to relevant application and error logs, each migration artifact directory also contains a `upload_target.zip` file which you can deploy to Elastic Beanstalk.

```
PS C:\migrations_workspace> ls .\migrations\latest\

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
```

```
d-----        3/18/2025  10:34 PM                upload_target
-a----        3/18/2025  10:34 PM          13137 application.log
-a----        3/18/2025  10:34 PM              0 error.log
-a----        3/18/2025  10:34 PM        1650642 upload_target.zip
```

You can deploy the `upload_target.zip` file using **eb migrate**:

```
PS C:\migrations_workspace> eb migrate --zip .\migrations\latest\upload_target.zip
```

Troubleshooting and diagnostics

This section provides guidance for troubleshooting common issues that may arise during the migration of IIS applications to Elastic Beanstalk.

Associating an EC2 keypair with your environment

You can securely log in to the Amazon Elastic Compute Cloud(Amazon EC2) instances provisioned for your Elastic Beanstalk application with an Amazon EC2 key pair. For instructions on creating a key pair, see Creating a Key Pair Using Amazon EC2 in the *Amazon EC2 User Guide*.

Specifying a keyname to **eb migrate** has the effect of associating your Elastic Beanstalk environment with the keypair. For security purposes, this will not open up port 3389 on your EC2 instances security group. You can associate additional EC2 security groups allowing traffic at port 3389 through **eb config** after the initial migration.

```
PS C:\migrations_workspace> eb migrate  `
    --keyname "my-keypair"  `
    --verbose
```

When you create a key pair, Amazon EC2 stores a copy of your public key. If you no longer need to use it to connect to any environment instances, you can delete it from Amazon EC2. For details, see Deleting Your Key Pair in the *Amazon EC2 User Guide*.

For more information about connecting to Windows Amazon EC2 instances, see Connecting to Windows Instance.

Accessing logs

The EB CLI provides an **eb logs** facility which you can use to retrieve logs from an Elastic Beanstalk environment without logging into its EC2 instances. After an execution of **eb**

migrate, you can issue the **eb logs --zip** command which will download and save logs into the `.elasticbeanstalk\logs` directory.

Alternatively, you can view logs through the AWS Elastic Beanstalk console. For more information, see Viewing logs from Amazon EC2 instances in your Elastic Beanstalk environment.

Accessing client-side artifacts

The **eb migrate** command stores application and error logs generated by **msdeploy** inside migrations artifacts directories.

```
./migrations/
### latest -> migration_20240308_123456/
### migration_20240308_123456/
    ### application.log
    ### error.log
    ### upload_target\
```

Monitoring environment health

Elastic Beanstalk helps you monitor health using the enhanced health monitoring capabilities. It is an automated health monitoring system that continuously tracks the operational status of application instances, leveraging built-in metrics such as CPU utilization, latency, request counts, and response codes.

The health monitoring system utilizes an agent-based approach to collect instance-level data and integrates with real-time logging and alerting. Elastic Load Balancing (ELB) and Auto Scaling dynamically respond to health status changes, ensuring high availability and fault tolerance. Advanced monitoring modes, including enhanced health reporting, provide granular visibility into application behavior, enabling proactive troubleshooting and automatic recovery mechanisms.

Run the EB CLI **eb health** command to display the environment's health. The following information displays:

- Instance health status
- Application response metrics
- System resource utilization
- Recent deployment events

EC2 performance optimization

By default, **eb migrate** selects the c5.2xlarge instance type to provide an optimal first-time experience with Elastic Beanstalk. You can override this behavior with the **--instance-type** argument:

```
PS C:\migrations_workspace> eb migrate `
    --instance-type "t3.large"
```

For production environments, consider these factors when selecting an instance type:

- Memory requirements of your applications
- CPU requirements for processing workloads
- Network performance needs
- Cost optimization goals

EBS volume configuration

By default, Elastic Beanstalk will create only a root block-device volume (`C:\`) for your environment. You can pass additional Amazon Elastic Block Store snapshot volumes with the **--ebs-snapshots** option:

```
PS C:\migrations_workspace> eb migrate `
    --ebs-snapshots "snap-123456789abc"
```

For examples of how you can configure block-device mappings with Elastic Beanstalk, see the blog article Customize Ephemeral and EBS Volumes in Elastic Beanstalk Environments.

For applications with high storage requirements, consider the following options:

- Using EBS volumes for persistent data
- Implementing Amazon S3 for static content
- Using Amazon FSx for Windows File Server for shared file systems

Common issues and solutions

Event: *Missing Web Deploy installation*

If you encounter errors related to Web Deploy not being found, then install Web Deploy 3.6 or later from the Microsoft Web Platform Installer. The following example displays a possible error message.

```
Couldn't find msdeploy.exe. Follow instructions here: https://learn.microsoft.com/en-us/iis/install/installing-publishing-technologies/installing-and-configuring-web-deploy
```

Event: *Permission issues during migration*

If you encounter permission-related errors, then ensure that you're running the EB CLI with administrative privileges. The following example displays a possible error message.

```
[ERROR] Access to the path 'C:\inetpub\wwwroot\web.config' is denied.
```

Event: *Application pool identity issues*

If your application fails to start due to application pool identity issues, create a custom script to configure application pool identities as shown in the section called "Custom application pool settings".

Event: *SSL certificate configuration errors*

If HTTPS bindings fail to work, ensure that you've specified a valid ACM certificate ARN using the **eb mibrate** option `--ssl-certificates` parameter.

Event: *Environment creation timeout*

If environment creation times out, check AWS CloudFormation events in the AWS Management Console for specific resource creation failures. Common causes include VPC configuration issues or service limits.

Getting support

If you encounter issues that you cannot resolve, before contacting AWS Support gather the following information:

- Environment ID (`eb status`)
- Application logs (`eb logs --zip`)
- Migration artifacts from `.\migrations\latest\`
- Source IIS configuration (output of `eb migrate explore --verbose`)

- Detailed error messages

For more information about Elastic Beanstalk troubleshooting, see Troubleshooting your Elastic Beanstalk environment.

Comparing migration options: EB CLI vs. AWS Application Migration Service

AWS offers multiple paths for migrating Windows applications to the cloud. This section compares two primary options: the **eb migrate** command in the EB CLI and AWS Application Migration Service (MGN). Understanding the differences between these approaches will help you choose the most appropriate migration strategy for your specific needs.

Comparison of migration options

Feature	EB CLI (eb migrate)	AWS Application Migration Service (MGN)
Primary focus	Application-level migration of IIS websites and applications	Server-level rehosting of entire machines (physical, virtual, or cloud servers)
Best suited for	IIS applications that you want to migrate directly to Elastic Beanstalk with minimal reconfiguration	Large-scale migrations involving many servers or complex infrastru cture
Discovery approach	Application-level discovery of IIS sites, applications, and configura tions	Server-level replication of entire machines, including operating system and applications
Target environment	Directly creates and configures Elastic Beanstalk environments optimized for Windows applications	Creates EC2 instances that require additional configuration to work with Elastic Beanstalk
Configuration preservation	Automatically preserves IIS-speci fic configurations (sites, application pools, bindings)	Preserves entire server configura tion, which may include unnecessary components

Feature	EB CLI (eb migrate)	AWS Application Migration Service (MGN)
Deployment model	Creates a clean Elastic Beanstalk environment with your applications deployed using Elastic Beanstalk best practices	Creates a replica of your source server that may require optimization for cloud operations
Scale of migration	Ideal for targeted migrations of specific applications	Designed for large-scale migrations of many servers
Post-migration steps	Minimal; environment is ready for use with Elastic Beanstalk management tools	Requires additional steps to integrate with Elastic Beanstalk, such as executing SSM post-launch actions

When to use each migration option

Choose eb migrate when you have the following requirements:

- You want to migrate specific IIS applications rather than entire servers
- Your goal is to adopt Elastic Beanstalk as your application management platform
- You want to leverage Elastic Beanstalk's managed platform features like easy scaling, deployment, and monitoring
- You prefer a clean deployment that follows AWS best practices for cloud-native operations
- You want to minimize post-migration configuration work

Choose AWS Application Migration Service when you have the following requirements:

- You need to migrate a large number of servers
- You have complex server configurations that must be preserved exactly
- Your applications have compatibility issues that require maintaining the exact server environment
- You want to "lift and shift" with minimal changes to your applications
- You plan to refactor or optimize your applications after migration

Migration workflow comparison

EB CLI (eb migrate) workflow:

1. Install the EB CLI on either your source IIS server or a bastion host.
2. Run **eb migrate** to discover IIS applications.
3. The command packages your applications and configurations.
4. An Elastic Beanstalk environment is created with appropriate resources.
5. Your applications are deployed to the new environment.
6. You can immediately manage your applications using Elastic Beanstalk tools.

AWS Application Migration Service workflow:

1. Install the AWS Replication Agent on source servers.
2. Configure and test data replication.
3. Launch test instances to verify functionality.
4. Schedule cutover to AWS.
5. Launch production instances.
6. Execute post-launch actions to optimize for cloud.
7. If Elastic Beanstalk is the target platform, additional configuration is required to integrate with Elastic Beanstalk.

Conclusion

Elastic Beanstalk is the preferred destination for Windows platform applications on AWS, offering a managed environment that simplifies deployment, scaling, and management. The **eb migrate** command provides a direct path to Elastic Beanstalk for IIS applications, with automatic discovery and configuration that preserves your application settings.

While AWS Application Migration Service offers powerful capabilities for large-scale server migrations, it requires additional steps to integrate with Elastic Beanstalk. For most IIS application migrations where Elastic Beanstalk is the target platform, **eb migrate** offers a more streamlined approach that aligns with Elastic Beanstalk's managed service model.

Choose the migration approach that best fits your specific requirements, considering factors such as scale, complexity, and your desired end-state architecture on AWS.

For more information about AWS Application Migration Service, see What is AWS Application Migration Service? in the AWS Application Migration Service User Guide.

Troubleshooting your Elastic Beanstalk environment

This chapter provides guidance for troubleshooting issues with your Elastic Beanstalk environment. It provides the following information.

- An introduction to the AWS Systems Manager tool, plus a procedure to run a predefined Elastic Beanstalk runbook that outputs troubleshooting steps and recommendations.
- General guidance for actions you can take and resources you can view if your environment status degrades.
- More specific troubleshooting tips by subject category.

Note

If the health of your environment changes to red, we recommend that you first use the AWS Systems Manager tool that includes predefined runbooks to troubleshoot Elastic Beanstalk. For more information see the Using the Systems Manager tool.

Topics

Using AWS Systems Manager Elastic Beanstalk runbooks

You can use Systems Manager to troubleshoot your Elastic Beanstalk environments. To help you get started quickly, Systems Manager provides predefined Automation runbooks for Elastic Beanstalk. An Automation runbook is a type of Systems Manager document that defines actions to perform on your environment's instances and other AWS resources.

The document `AWSSupport-TroubleshootElasticBeanstalk` is an Automation runbook designed to help identify a number of common issues that can degrade your Elastic Beanstalk environment. To do so, it checks components of your environment, including the following: EC2 instances, the VPC, AWS CloudFormation stack, load balancers, Auto Scaling groups, and network configuration associated with security group rules, route tables, and ACLs.

It also provides an option to upload bundled log files from your environment to AWS Support.

For more information, see `AWSSupport-TroubleshootElasticBeanstalk` in the *AWS Systems Manager Automation runbook reference*.

Use Systems Manager to run `AWSSupport-TroubleshootElasticBeanstalk` runbook

Note

Run this procedure in the same AWS Region where your Elastic Beanstalk environment is located.

1. Open the AWS Systems Manager console.
2. From the navigation pane, in the **Change Management** section, choose **Automation**.
3. Choose **Execute automation**.
4. On the **Owned by Amazon** tab, in the **Automation document** search box, enter `AWSSupport-TroubleshootElasticBeanstalk`.
5. Select the **AWSSupport-TroubleshootElasticBeanstalk** card, then choose **Next**.
6. Select **Execute**.
7. In the **Input parameters** section:

    a. From the **AutomationAssumeRole** dropdown, select the ARN of the role that allows Systems Manager to perform actions on your behalf.

    b. For **ApplicationName**, enter the name of the Elastic Beanstalk application.

   c. For **Environment Name**, enter the Elastic Beanstalk environment.
   d. (Optional) For **S3UploaderLink**, enter a link if an AWS Support Engineer has provided you an S3 link for log collection.

8. Choose **Execute**.

   If any of the steps fail, select the link under the **Step ID** column for the step that failed. This displays an **Execution detail** page for the step. The **VerificationErrorMessage** section will display a summary of the steps that require attention. For example, the `IAMPermissionCheck` could display a Warning message. In this case, you could check that the role selected in the **AutomationAssumeRole** dropdown has the necessary permissions.

After all of the steps successfully complete, the output gives troubleshooting steps and recommendations to restore your environment to a healthy state.

General guidance for troubleshooting your Elastic Beanstalk environment

Error messages can appear on the Events page in the console, in logs, or on the Health page. You can also take actions to recover from a degraded environment that was caused by a recent change. If the health of your environment changes to Red, try the following:

- If an operation on your environment returns an error that contains the text `The stack *stack_id* associated with environment *environment-ID* is in *stack-status* state`, see Recovering your Elastic Beanstalk environment from an invalid state for troubleshooting help.
- If an operation on your environment returns an error that contains the text `Environment *environment-name* associated CloudFormation stack *stack_arn* does not exist`, terminate your environment and create another one.
- Review recent environment events. Messages from Elastic Beanstalk about deployment, load, and configuration issues often appear here.
- Review recent environment change history. Change history lists all of the configuration changes made to your environments and includes other information, such as which IAM user made changes and which configuration parameters were set.
- Pull logs to view recent log file entries. Web server logs contain information about incoming requests and errors.

- Connect to an instance and check system resources.
- Roll back to a previous working version of the application.
- Undo recent configuration changes or restore a saved configuration.
- Deploy a new environment. If the environment appears healthy, perform a CNAME swap to route traffic to the new environment and continue to debug the previous one.

Environments that access secrets and parameters with environment variables

Event: *Instance deployment failed to get one or more secrets*

This message indicates that Elastic Beanstalk was not able to fetch one or more of the secrets specified during your application deployment.

- Check that the resources specified by the ARN values in your environment variable configuration exist.
- Confirm that your Elastic Beanstalk EC2 instance profile role has the required IAM permissions to access the resources.
- If this event was triggered through the `RestartAppServer` operation, once the issue is fixed, retry the `RestartAppServer` call to resolve the issue.
- If the event was triggered through an `UpdateEnvironment` call, retry the `UpdateEnvironment` operation.

For examples of these commands, see *AWS CLI examples for Elastic Beanstalk*. For more information about the API actions for these operations, see the *AWS Elastic Beanstalk API Reference*.

Event: *Instance deployment detected one or more multiline environment values, which are not supported for this platform*

Multiline variables are not supported for Amazon Linux 2 platforms, excluding Docker and ECS managed Docker platforms. For available options to proceed, see Multiline values.

Event: *CreateEnvironment fails when a secret is specified*

When `CreateEnvironment` fails and you have secrets as environment variables, you need to address the underlying issue and then use `UpdateEnvironment` to complete the environment setup. Do not use `RestartAppServer`, as it will not be sufficient to bring the environment up in this situation. For examples of these commands, see *AWS CLI examples for Elastic Beanstalk*. For more information about the API actions for these operations, see the *AWS Elastic Beanstalk API Reference*.

Environment creation and instance launches

Event: *Failed to Launch Environment*

This event occurs when Elastic Beanstalk attempts to launch an environment and encounters failures along the way. Previous events on the **Events** page will alert you to the root cause of this issue.

Event: *Create environment operation is complete, but with command timeouts. Try increasing the timeout period.*

Your application may take a long time to deploy if you use configuration files that run commands on the instance, download large files, or install packages. Increase the command timeout to give your application more time to start running during deployments.

Event: *The following resource(s) failed to create: [AWSEBInstanceLaunchWaitCondition]*

This message indicates that your environment's Amazon EC2 instances did not communicate to Elastic Beanstalk that they were launched successfully. This can occur if the instances do not have Internet connectivity. If you configured your environment to launch instances in a private VPC subnet, ensure that the subnet has a NAT to allow the instances to connect to Elastic Beanstalk.

Event: *A Service Role is required in this region. Please add a Service Role option to the environment.*

Elastic Beanstalk uses a service role to monitor the resources in your environment and support managed platform updates. See Managing Elastic Beanstalk service roles for more information.

Deployments

Issue: *Application becomes unavailable during deployments*

Because Elastic Beanstalk uses a drop-in upgrade process, there might be a few seconds of downtime. Use rolling deployments to minimize the effect of deployments on your production environments.

Event: *Failed to create the AWS Elastic Beanstalk application version*

Your application source bundle may be too large, or you may have reached the application version quota.

Event: *Update environment operation is complete, but with command timeouts. Try increasing the timeout period.*

Your application may take a long time to deploy if you use configuration files that run commands on the instance, download large files, or install packages. Increase the command timeout to give your application more time to start running during deployments.

Health

Event: *CPU Utilization Exceeds 95.00%*

Try running more instances, or choose a different instance type.

Event: *Elastic Load Balancer awseb-`myapp` Has Zero Healthy Instances*

If your application appears to be working, make sure that your application's health check URL is configured correctly. If not, check the Health screen and environment logs for more information.

Event: *Elastic Load Balancer awseb-`myapp` Cannot Be Found*

Your environment's load balancer may have been removed out-of-band. Only make changes to your environment's resources with the configuration options and extensibility provided by Elastic Beanstalk. Rebuild your environment or launch a new one.

Event: *EC2 Instance Launch Failure. Waiting for a New EC2 Instance to Launch...*

Availability for your environment's instance type may be low, or you may have reached the instance quota for your account. Check the service health dashboard to ensure that the Elastic Compute Cloud (Amazon EC2) service is green, or request a quota increase.

Configuration

Event: *The stack `stack_id` associated with environment `environment-ID` is in `stack-status` state*

The underlying AWS CloudFormation stack of your environment may be in a **_FAILED* status. This status must be remedied in order to continue Elastic Beanstalk operations on your environment. For more information, see Recovering your Elastic Beanstalk environment from an invalid state.

Event: *You cannot configure an Elastic Beanstalk environment with values for both the Elastic Load Balancing Target option and Application Healthcheck URL option*

The `Target` option in the `aws:elb:healthcheck` namespace is deprecated. Remove the `Target` option namespace) from your environment and try updating again.

Event: *ELB cannot be attached to multiple subnets in the same AZ*

This message can be seen if you try to move a load balancer between subnets in the same Availability Zone. Changing subnets on the load balancer requires moving it out of the original availability zone(s) and then back into the original with the desired subnets. During the process, all of your instances will be migrated between AZs, causing significant downtime. Instead, consider creating a new environment and perform a CNAME swap.

Troubleshooting Docker containers

Event: *Failed to pull Docker image :latest: Invalid repository name (), only [a-z0-9-_.] are allowed. Tail the logs for more details.*

Check the syntax of the `dockerrun.aws.json` file using a JSON validator. Also verify the dockerfile contents against the requirements described in Preparing your Docker image for deployment to Elastic Beanstalk

Event: *No EXPOSE directive found in Dockerfile, abort deployment*

The `Dockerfile` or the `dockerrun.aws.json` file does not declare the container port. Use the `EXPOSE` instruction (`Dockerfile`) or `Ports` block (`dockerrun.aws.json` file) to expose a port for incoming traffic.

Event: *Failed to download authentication credentials `repository` from `bucket name`*

The `dockerrun.aws.json` provides an invalid EC2 key pair and/or S3 bucket for the `.dockercfg` file. Or, the instance profile does not have GetObject authorization for the S3 bucket. Verify that the `.dockercfg` file contains a valid S3 bucket and EC2 key pair. Grant permissions for the action `s3:GetObject` to the IAM role in the instance profile. For details, go to Managing Elastic Beanstalk instance profiles

Event: *Activity execution failed, because: WARNING: Invalid auth configuration file*

Your authentication file (`config.json`) is not formatted correctly. See Authenticating with image repositories

FAQ

Question: *How can I change my application URL from myapp.us-west-2.elasticbeanstalk.com to www.myapp.com?*

In a DNS server, register a CNAME record such as **`www.mydomain.com CNAME mydomain.elasticbeanstalk.com`**.

Question: *How do I specify a specific Availability Zone for my Elastic Beanstalk application?*

You can pick a specific Availability Zone by using the APIs, CLI, Eclipse plugin, or Visual Studio plugin. For instructions about using the Elastic Beanstalk console to specify an Availability Zone, see Auto Scaling your Elastic Beanstalk environment instances.

Question: *How do I change my environment's instance type?*

To change your environment's instance type go to the environment configuration page and choose **Edit** in the **Instances** configuration category. Then, select a new instance type and choose **Apply** to update your environment. After this, Elastic Beanstalk terminates all running instances and replaces them with new ones.

Question: *How do I determine if anyone made configuration changes to an environment?*

To see this information, in the navigation pane of the Elastic Beanstalk console choose **Change history** to display a list of configuration changes for all environments. This list includes the date and time of the change, the configuration parameter and value it was changed to, and the IAM user that made the change. For more information, see Change history.

Question: *Can I prevent Amazon EBS volumes from being deleted when instances are terminated?*

Instances in your environment use Amazon EBS for storage; however, the root volume is deleted when an instance is terminated by Auto Scaling. We don'trecommend that you store state or other data on your instances. If needed, you can prevent volumes from being deleted with the AWS CLI: `$ aws ec2 modify-instance-attribute -b '/dev/sdc=<vol-id>:false` as described in the AWS CLI Reference.

Question: *How do I delete personal information from my Elastic Beanstalk application?*

AWS resources that your Elastic Beanstalk application uses might store personal information. When you terminate an environment, Elastic Beanstalk terminates the resources that it created. Resources you added using configuration files are also terminated. However, if you created AWS resources outside of your Elastic Beanstalk environment and associated them with your application, you might need to manually check that personal information that your application might have stored isn't retained. Throughout this developer guide, whenever we discuss creating additional resources, we also mention when you should consider deleting them.

Common Errors

This topic lists common error messages encountered when using the EB CLI and possible solutions. If you encounter an error message not shown here, use the *Feedback* links to let us know about it.

ERROR: An error occurred while handling git command. Error code: 128 Error: fatal: Not a valid object name HEAD

Cause: This error message is shown when you have initialized a Git repository but have not yet committed. The EB CLI looks for the HEAD revision when your project folder contains a Git repository.

Solution: Add the files in your project folder to the staging area and commit:

```
~/my-app$ git add .
~/my-app$ git commit -m "First commit"
```

ERROR: This branch does not have a default environment. You must either specify an environment by typing "eb status my-env-name" or set a default environment by typing "eb use my-env-name".

Cause: When you create a new branch in git, it is not attached to an Elastic Beanstalk environment by default.

Solution: Run **eb list** to see a list of available environments. Then run **eb use *env-name*** to use one of the available environments.

ERROR: 2.0+ Platforms require a service role. You can provide one with --service-role option

Cause: If you specify an environment name with **eb create** (for example, **eb create my-env**), the EB CLI will not attempt to create a service role for you. If you don't have the default service role, the above error is shown.

Solution: Run **eb create** without an environment name and follow the prompts to create the default service role.

Deployment errors

Your Elastic Beanstalk deployment might response with a 404 (if your application failed to launch) or 500 (if your application fails during runtime) response. To troubleshoot many common issues, you can use the EB CLI to check the status of your deployment, view its logs, gain access to your EC2 instance with SSH, or to open the AWS Management Console page for your application environment.

To use the EB CLI to help troubleshoot your deployment

1. Run **eb status** to see the status of your current deployment and health of your EC2 hosts. For example:

```
$ eb status --verbose

Environment details for: python_eb_app
  Application name: python_eb_app
  Region: us-west-2
  Deployed Version: app-150206_035343
  Environment ID: e-wa8u6rrmqy
  Platform: 64bit Amazon Linux 2014.09 v1.1.0 running Python 2.7
  Tier: WebServer-Standard-
  CNAME: python_eb_app.elasticbeanstalk.com
  Updated: 2015-02-06 12:00:08.557000+00:00
  Status: Ready
  Health: Green
  Running instances: 1
      i-8000528c: InService
```

> **Note**
>
> Using the `--verbose` switch provides information about the status of your running instances. Without it, **eb status** will print only general information about your environment.

2. Run **eb health** to view health information about your environment:

```
$ eb health --refresh
 elasticBeanstalkExa-env                                          Degraded
 2016-03-28 23:13:20
```

```
WebServer
    Ruby 2.1 (Puma)
  total      ok    warning  degraded  severe    info   pending  unknown
    5        2        0        2        1        0        0        0

  instance-id   status     cause
    Overall     Degraded  Incorrect application version found on 3 out of 5
 instances. Expected version "Sample Application" (deployment 1).
  i-d581497d    Degraded  Incorrect application version "v2" (deployment 2).
 Expected version "Sample Application" (deployment 1).
  i-d481497c    Degraded  Incorrect application version "v2" (deployment 2).
 Expected version "Sample Application" (deployment 1).
  i-136e00c0    Severe    Instance ELB health has not been available for 5 minutes.
  i-126e00c1    Ok
  i-8b2cf575    Ok

  instance-id   r/sec    %2xx   %3xx   %4xx   %5xx      p99      p90      p75
 p50     p10
    Overall     646.7   100.0    0.0    0.0    0.0    0.003    0.002    0.001
 0.001   0.000
  i-dac3f859    167.5    1675      0      0      0    0.003    0.002    0.001
 0.001   0.000
  i-05013a81    161.2    1612      0      0      0    0.003    0.002    0.001
 0.001   0.000
  i-04013a80    0.0         -      -      -      -        -        -       -
 -       -
  i-3ab524a1    155.9    1559      0      0      0    0.003    0.002    0.001
 0.001   0.000
  i-bf300d3c    162.1    1621      0      0      0    0.003    0.002    0.001
 0.001   0.000

  instance-id   type       az   running     load 1  load 5      user%  nice%
 system%  idle%   iowait%
  i-d581497d    t2.micro   1a   25 mins       0.16     0.1        7.0    0.0
 1.7   91.0       0.1
  i-d481497c    t2.micro   1a   25 mins       0.14     0.1        7.2    0.0
 1.6   91.1       0.0
  i-136e00c0    t2.micro   1b   25 mins        0.0    0.01        0.0    0.0
 0.0   99.9       0.1
  i-126e00c1    t2.micro   1b   25 mins       0.03    0.08        6.9    0.0
 2.1   90.7       0.1
  i-8b2cf575    t2.micro   1c   1 hour        0.05    0.41        6.9    0.0
 2.0   90.9       0.0
```

```
  instance-id   status     id   version              ago
       deployments
  i-d581497d    Deployed   2    v2                   9 mins
  i-d481497c    Deployed   2    v2                   7 mins
  i-136e00c0    Failed     2    v2                   5 mins
  i-126e00c1    Deployed   1    Sample Application   25 mins
  i-8b2cf575    Deployed   1    Sample Application   1 hour
```

The above example shows an environment with five instances where the deployment of version "v2" failed on the third instance. After a failed deployment, the expected version is reset to the last version that succeeded, which in this case is "Sample Application" from the first deployment. See Using the EB CLI to monitor environment health for more information.

3. Run **eb logs** to download and view the logs associated with your application deployment.

```
$ eb logs
```

4. Run **eb ssh** to connect with the EC2 instance that's running your application and examine it directly. On the instance, your deployed application can be found in the `/opt/python/current/app` directory, and your Python environment will be found in `/opt/python/run/venv/`.
5. Run **eb console** to view your application environment on the AWS Management Console. You can use the web interface to easily examine various aspects of your deployment, including your application's configuration, status, events, logs. You can also download the current or past application versions that you've deployed to the server.

Elastic Beanstalk resources

The following related resources can help you as you work with this service.

- **Elastic Beanstalk API Reference** A comprehensive description of all SOAP and Query APIs. Additionally, it contains a list of all SOAP data types.
- **elastic-beanstalk-samples on GitHub** – A GitHub repository with Elastic Beanstalk sample configuration files (.ebextensions). The repository's `README.md` file has links to additional GitHub repositories with sample applications.
- **Elastic Beanstalk Technical FAQ** – The top questions developers have asked about this product.
- **AWS Elastic Beanstalk Release Notes** – Details about new features, updates, and fixes in Elastic Beanstalk service, platform, console, and EB CLI releases.

- Classes & Workshops – Links to role-based and specialty courses, in addition to self-paced labs to help sharpen your AWS skills and gain practical experience.
- AWS Developer Center – Explore tutorials, download tools, and learn about AWS developer events.
- AWS Developer Tools – Links to developer tools, SDKs, IDE toolkits, and command line tools for developing and managing AWS applications.
- Getting Started Resource Center – Learn how to set up your AWS account, join the AWS community, and launch your first application.
- Hands-On Tutorials – Follow step-by-step tutorials to launch your first application on AWS.
- AWS Whitepapers – Links to a comprehensive list of technical AWS whitepapers, covering topics such as architecture, security, and economics and authored by AWS Solutions Architects or other technical experts.
- AWS Support Center – The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.
- Support – The primary webpage for information about Support, a one-on-one, fast-response support channel to help you build and run applications in the cloud.
- Contact Us – A central contact point for inquiries concerning AWS billing, account, events, abuse, and other issues.

- AWS Site Terms – Detailed information about our copyright and trademark; your account, license, and site access; and other topics.

Sample applications

The following are download links to the sample applications that are deployed as part of Learn how to get started with Elastic Beanstalk.

Note

Some samples use features that may have been released since the release of the platform you are using. If the sample fails to run, try updating your platform to a current version, as described in the section called "Supported platforms".

- **Docker** – docker.zip
- **Multicontainer Docker** – docker-multicontainer-v2.zip
- **Preconfigured Docker (Glassfish)** – docker-glassfish-v1.zip
- **Go** – go.zip
- **Corretto** – corretto.zip
- **Tomcat** – tomcat.zip
- **.NET Core on Linux** – dotnet-core-linux.zip
- **.NET Core** – dotnet-asp-windows.zip
- **Node.js** – nodejs.zip
- **PHP** – php.zip
- **Python** – python.zip
- **Ruby** – ruby.zip

AWS SDK for Java

The AWS SDK for Java provides a Java API you can use to build applications that use AWS infrastructure services. With the AWS SDK for Java, you can get started in minutes with a single, downloadable package that includes the AWS Java library, code examples, and documentation.

The AWS SDK for Java requires the J2SE Development Kit. You can download the latest Java software from http://developers.sun.com/downloads/. The SDK also requires Apache Commons (Codec, HTTPClient, and Logging) and Saxon-HE third-party packages, which are included in the third-party directory of the SDK.

For more information, see AWS SDK for Java. For more information about the SDK, example code, documentation, tools, and additional resources, see the Java on AWS Developer Center.

AWS SDK for .NET

With the AWS SDK for .NET, you can get started quickly with a single, downloadable package that includes the AWS .NET library, code examples, and documentation.

For more information, see AWS SDK for .NET. For supported .NET Framework and Visual Studio versions, see the AWS SDK for .NET Developer Guide.

For more information about the SDK, example code, documentation, tools, and additional resources, see the .NET on AWS Developer Center.

AWS Toolkit for Visual Studio

With the AWS Toolkit for Visual Studio plug-in, you can deploy an existing .NET application to Elastic Beanstalk. You can also create projects using the AWS templates that are preconfigured with the AWS SDK for .NET.

For prerequisite and installation information, see the AWS Toolkit for Visual Studio. To get started creating your Elastic Beanstalk application using Visual Studio, see Deploying .NET Windows applications with Elastic Beanstalk.

AWS SDK for JavaScript in Node.js

With the AWS SDK for JavaScript in Node.js, you can get started quickly with a single, downloadable package that includes the AWS Node.js library, code examples, and documentation.

For more information, see the AWS SDK for JavaScript in Node.js.

AWS SDK for PHP

With the AWS SDK for PHP, you can get started quickly with a single, downloadable package that includes the AWS PHP library, code examples, and documentation.

For more information, see the AWS SDK for PHP. For more information about the SDK, example code, documentation, tools, and additional resources, see the PHP on AWS Developer Center.

AWS SDK for Python (Boto)

With the AWS SDK for Python (Boto), you can get started quickly with a single, downloadable package that includes the AWS Python library, code examples, and documentation. You can build Python applications on top of APIs that take the complexity out of coding directly against web service interfaces.

The all-in-one library provides Python developer-friendly APIs that hide many of the lower-level tasks associated with programming for the AWS Cloud, including authentication, request retries, and error handling. The SDK provides practical examples in Python for how to use the libraries to build applications.

For information about Boto, example code, documentation, tools, and additional resources, see the Python on AWS Developer Center.

AWS SDK for Ruby

With the AWS Ruby library, code examples, and documentation, you can build Ruby applications on top of APIs that take the complexity out of coding directly against web services interfaces.

The all-in-one library provides Ruby developer-friendly APIs that hide many of the lower-level tasks associated with programming for the AWS Cloud, including authentication, request retries, and error handling. The SDK provides practical examples in Ruby for how to use the libraries to build applications.

For information about the SDK, example code, documentation, tools, and additional resources, see the Ruby on AWS Developer Center.

Archived content for the AWS Elastic Beanstalk Developer Guide

For reference purposes, this chapter provides configuration information about some Elastic Beanstalk legacy features.

Topics

Elastic Beanstalk custom platforms (retired)

Note

On July 18, 2022, Elastic Beanstalk set the status of all platform branches based on Amazon Linux AMI (AL1) to **retired**. This includes custom platforms. Elastic Beanstalk doesn't support custom platforms. For more information about Elastic Beanstalk's retirement of Amazon Linux AMI, see Platform retirement FAQ.

This topic remains in this document as a reference for any customers that used the Elastic Beanstalk custom platform feature prior to its retirement. In the past, Elastic Beanstalk custom platforms supported building an AMI from Amazon Linux AMI, RHEL 7, RHEL 6, or Ubuntu 16.04 base AMIs. These operating systems are no longer supported by Elastic Beanstalk. To read more about the custom platforms feature, which is no longer supported, expand the following topic.

Custom platforms

A custom platform is a more advanced customization than a custom image in several ways. A custom platform lets you develop an entire new platform from scratch, customizing the operating system, additional software, and scripts that Elastic Beanstalk runs on platform instances. This flexibility enables you to build a platform for an application that uses a language or other infrastructure software, for which Elastic Beanstalk doesn't provide a managed platform. Compare that to custom images, where you modify an Amazon Machine Image (AMI) for use with an existing Elastic Beanstalk platform, and Elastic Beanstalk still provides the platform scripts and controls the platform's software stack. In addition, with custom platforms you use an automated, scripted way to create and maintain your customization, whereas with custom images you make the changes manually over a running instance.

To create a custom platform, you build an AMI from one of the supported operating systems—Ubuntu, RHEL, or Amazon Linux (see the `flavor` entry in Platform.yaml file format for the exact version numbers)—and add further customizations. You create your own Elastic Beanstalk platform using Packer, which is an open-source tool for creating machine images for many platforms, including AMIs for use with Amazon Elastic Compute Cloud (Amazon EC2). An Elastic Beanstalk platform comprises an AMI configured to run a set of software that supports an application, and metadata that can include custom configuration options and default configuration option settings.

Elastic Beanstalk manages Packer as a separate built-in platform, and you don't need to worry about Packer configuration and versions.

You create a platform by providing Elastic Beanstalk with a Packer template, and the scripts and files that the template invokes to build an AMI. These components are packaged with a platform definition file, which specifies the template and metadata, into a ZIP archive, known as a platform definition archive.

When you create a custom platform, you launch a single instance environment without an Elastic IP that runs Packer. Packer then launches another instance to build an image. You can reuse this environment for multiple platforms and multiple versions of each platform.

Note

Custom platforms are AWS Region specific. If you use Elastic Beanstalk in multiple Regions, you must create your platforms separately in each Region.
In certain circumstances, instances launched by Packer are not cleaned up and have to be manually terminated. To learn how to manually clean up these instances, see Packer instance cleanup.

Users in your account can use your custom platforms by specifying a platform ARN during environment creation. These ARNs are returned by the **eb platform create** command that you used to create the custom platform.

Each time you build your custom platform, Elastic Beanstalk creates a new platform version. Users can specify a platform by name to get only the latest version of the platform, or include a version number to get a specific version.

For example, to deploy the latest version of the custom platform with the ARN **`MyCustomPlatformARN`**, which could be version 3.0, your EB CLI command line would look like this:

```
eb create -p MyCustomPlatformARN
```

To deploy version 2.1 your EB CLI command line would look like this:

```
eb create -p MyCustomPlatformARN --version 2.1
```

You can apply tags to a custom platform version when you create it, and edit tags of existing custom platform versions. For details, see Tagging custom platform versions.

Creating a custom platform

To create a custom platform, the root of your application must include a platform definition file, `platform.yaml`, which defines the type of builder used to create the custom platform. The format of this file is described in Platform.yaml file format. You can create your custom platform from scratch, or use one of the sample custom platforms as a starting point.

Using a sample custom platform

One alternative to creating your own custom platform is to use one of the platform definition archive samples to bootstrap your custom platform. The only items you have to configure in the samples before you can use them are a source AMI and a Region.

Note

Do not use an unmodified sample custom platform in production. The goal of the samples is to show some of the functionality available for a custom platform, but they have not been hardened for production use.

NodePlatform_Ubuntu.zip

This custom platform is based on **Ubuntu 16.04** and supports **Node.js 4.4.4**. We use this custom platform for the examples in this section.

NodePlatform_RHEL.zip

This custom platform is based on **RHEL 7.2** and supports **Node.js 4.4.4**.

NodePlatform_AmazonLinux.zip

This custom platform is based on **Amazon Linux 2016.09.1** and supports **Node.js 4.4.4**.

TomcatPlatform_Ubuntu.zip

This custom platform is based on **Ubuntu 16.04** and supports **Tomcat 7/Java 8**.

CustomPlatform_NodeSampleApp.zip

A Node.js sample that uses **express** and **ejs** to display a static webpage.

CustomPlatform_TomcatSampleApp.zip

A Tomcat sample that displays a static webpage when deployed.

Download the sample platform definition archive: `NodePlatform_Ubuntu.zip`. This file contains a platform definition file, Packer template, scripts that Packer runs during image creation, and scripts and configuration files that Packer copies onto the builder instance during platform creation.

Example NodePlatform_Ubuntu.zip

```
|-- builder                 Contains files used by Packer to create the custom platform
|-- custom_platform.json    Packer template
|-- platform.yaml           Platform definition file
|-- ReadMe.txt              Briefly describes the sample
```

The platform definition file, `platform.yaml`, tells Elastic Beanstalk the name of the Packer template, `custom_platform.json`.

```
version: "1.0"

provisioner:
  type: packer
  template: custom_platform.json
  flavor: ubuntu1604
```

The Packer template tells Packer how to build the AMIs for the platform, using an Ubuntu AMI as a base for the platform image for HVM instance types. The `provisioners` section tells Packer to copy all files in the `builder` folder within the archive to the instance, and to run the `builder.sh` script on the instance. When the scripts complete, Packer creates an image from the modified instance.

Elastic Beanstalk creates three environment variables that can be used to tag AMIs in Packer:

AWS_EB_PLATFORM_ARN

The ARN of the custom platform.

AWS_EB_PLATFORM_NAME

The name of the custom platform.

AWS_EB_PLATFORM_VERSION

The version of the custom platform.

The sample `custom_platform.json` file uses these variables to define the following values that it uses in the scripts:

- `platform_name`, which is set by `platform.yaml`
- `platform_version`, which is set by `platform.yaml`
- `platform_arn`, which is set by the main build script, `builder.sh`, which is shown at the end of the sample `custom_platform.json` file.

The `custom_platform.json` file contains two properties that you have to provide values for: `source_ami` and `region`. For details about choosing the right AMI and Region values, see Updating Packer template in the *eb-custom-platforms-samples* GitHub repository.

Example custom_platform.json

```
{
  "variables": {
    "platform_name": "{{env `AWS_EB_PLATFORM_NAME`}}",
    "platform_version": "{{env `AWS_EB_PLATFORM_VERSION`}}",
    "platform_arn": "{{env `AWS_EB_PLATFORM_ARN`}}"
  },
  "builders": [
    {
      ...
      "region": "",
      "source_ami": "",
      ...
    }
  ],
  "provisioners": [
    {...},
    {
      "type": "shell",
      "execute_command": "chmod +x {{ .Path }}; {{ .Vars }} sudo {{ .Path }}",
      "scripts": [
        "builder/builder.sh"
      ]
    }
  ]
}
```

The scripts and other files that you include in your platform definition archive will vary greatly depending on the modifications that you want to make to the instance. The sample platform includes the following scripts:

- `00-sync-apt.sh` – Commented out: `apt -y update`. We commented out the command because it prompts the user for input, which breaks the automated package update. This might be an Ubuntu issue. However, running `apt -y update` is still recommended as a best practice. For this reason, we left the command in the sample script for reference.
- `01-install-nginx.sh` – Installs nginx.
- `02-setup-platform.sh` – Installs `wget`, `tree`, and `git`. Copies hooks and logging configurations to the instance, and creates the following directories:
  - `/etc/SampleNodePlatform` – Where the container configuration file is uploaded during deployment.
  - `/opt/elasticbeanstalk/deploy/appsource/` – Where the `00-unzip.sh` script uploads application source code during deployment (see the Platform script tools for your Elastic Beanstalk environments section for information about this script).
  - `/var/app/staging/` – Where application source code is processed during deployment.
  - `/var/app/current/` – Where application source code runs after processing.
  - `/var/log/nginx/healthd/` – Where the enhanced health agent writes logs.
  - `/var/nodejs` – Where the Node.js files are uploaded during deployment.

Use the EB CLI to create your first custom platform with the sample platform definition archive.

To create a custom platform

1. Install the EB CLI.
2. Create a directory in which you will extract the sample custom platform.

   ```
   ~$ mkdir ~/custom-platform
   ```

3. Extract `NodePlatform_Ubuntu.zip` to the directory, and then change to the extracted directory.

   ```
   ~$ cd ~/custom-platform
   ~/custom-platform$ unzip ~/NodePlatform_Ubuntu.zip
   ~/custom-platform$ cd NodePlatform_Ubuntu
   ```

4. Edit the `custom_platform.json` file, and provide values for the `source_ami` and `region` properties. For details, see Updating Packer template.
5. Run **eb platform init** and follow the prompts to initialize a platform repository.

   You can shorten **eb platform** to **ebp**.

   Note

   Windows PowerShell uses **ebp** as a command alias. If you're running the EB CLI in Windows PowerShell, use the long form of this command: **eb platform**.

   ```
   ~/custom-platform$ eb platform init
   ```

   This command also creates the directory `.elasticbeanstalk` in the current directory and adds the configuration file `config.yml` to the directory. Don't change or delete this file, because Elastic Beanstalk relies on it when creating the custom platform.

   By default, **eb platform init** uses the name of the current folder as the name of the custom platform, which would be `custom-platform` in this example.

6. Run **eb platform create** to launch a Packer environment and get the ARN of the custom platform. You'll need this value later when you create an environment from the custom platform.

```
~/custom-platform$ eb platform create
...
```

By default, Elastic Beanstalk creates the instance profile `aws-elasticbeanstalk-custom-platform-ec2-role` for custom platforms. If, instead, you want to use an existing instance profile, add the option `-ip` *`INSTANCE_PROFILE`* to the **eb platform create** command.

> **Note**
>
> Packer will fail to create a custom platform if you use the Elastic Beanstalk default instance profile `aws-elasticbeanstalk-ec2-role`.

The EB CLI shows event output of the Packer environment until the build is complete. You can exit the event view by pressing **Ctrl+C**.

7. You can check the logs for errors using the **eb platform logs** command.

```
~/custom-platform$ eb platform logs
...
```

8. You can check on the process later with **eb platform events**.

```
~/custom-platform$ eb platform events
...
```

9. Check the status of your platform with **eb platform status**.

```
~/custom-platform$ eb platform status
...
```

When the operation completes, you have a platform that you can use to launch an Elastic Beanstalk environment.

You can use the custom platform when creating an environment from the console. See The create new environment wizard.

To launch an environment on your custom platform

1. Create a directory for your application.

```
~$ mkdir custom-platform-app
~$ cd ~/custom-platform-app
```

2. Initialize an application repository.

```
~/custom-platform-app$ eb init
...
```

3. Download the sample application NodeSampleApp.zip.

4. Extract the sample application.

```
~/custom-platform-app$ unzip ~/NodeSampleApp.zip
```

5. Run **eb create -p *CUSTOM-PLATFORM-ARN***, where *CUSTOM-PLATFORM-ARN* is the ARN returned by an **eb platform create** command, to launch an environment running your custom platform.

```
~/custom-platform-app$ eb create -p CUSTOM-PLATFORM-ARN
...
```

Platform definition archive contents

A platform definition archive is the platform equivalent of an application source bundle. The platform definition archive is a ZIP file that contains a platform definition file, a Packer template, and the scripts and files used by the Packer template to create your platform.

Note

When you use the EB CLI to create a custom platform, the EB CLI creates a platform definition archive from the files and folders in your platform repository, so you don't need to create the archive manually.

The platform definition file is a YAML-formatted file that must be named `platform.yaml` and be in the root of your platform definition archive. See Creating a custom platform for a list of required and optional keys supported in a platform definition file.

You don't need to name the Packer template in a specific way, but the name of the file must match the provisioner template specified in the platform definition file. See the official Packer documentation for instructions on creating Packer templates.

The other files in your platform definition archive are scripts and files used by the template to customize an instance before creating an AMI.

Custom platform hooks

Elastic Beanstalk uses a standardized directory structure for hooks on custom platforms. These are scripts that are run during lifecycle events and in response to management operations: when instances in your environment are launched, or when a user initiates a deployment or uses the restart application server feature.

Place scripts that you want hooks to trigger in one of the subfolders of the `/opt/elasticbeanstalk/hooks/` folder.

Warning

Using custom platform hooks on managed platforms isn't supported. Custom platform hooks are designed for custom platforms. On Elastic Beanstalk managed platforms they might work differently or have some issues, and behavior might differ across platforms. On Amazon Linux AMI platforms (preceding Amazon Linux 2), they might still work in useful ways in some cases; use them with caution.
Custom platform hooks are a legacy feature that exists on Amazon Linux AMI platforms. On Amazon Linux 2 platforms, custom platform hooks in the `/opt/elasticbeanstalk/hooks/` folder are entirely discontinued. Elastic Beanstalk doesn't read or execute them. Amazon Linux 2 platforms support a new kind of platform hooks, specifically designed to extend Elastic Beanstalk managed platforms. You can add custom scripts and programs directly to a hooks directory in your application source bundle. Elastic Beanstalk runs them during various instance provisioning stages. For more information, expand the *Platform Hooks* section in the section called “Extending Linux platforms”.

Hooks are organized into the following folders:

- `appdeploy` — Scripts run during an application deployment. Elastic Beanstalk performs an application deployment when new instances are launched and when a client initiates a new version deployment.
- `configdeploy` — Scripts run when a client performs a configuration update that affects the software configuration on instance, for example, by setting environment properties or enabling log rotation to Amazon S3.
- `restartappserver` — Scripts run when a client performs a restart app server operation.
- `preinit` — Scripts run during instance bootstrapping.
- `postinit` — Scripts run after instance bootstrapping.

The `appdeploy`, `configdeploy`, and `restartappserver` folders contain `pre`, `enact`, and `post` subfolders. In each phase of an operation, all scripts in the `pre` folder are run in alphabetical order, then those in the `enact` folder, and then those in the `post` folder.

When an instance is launched, Elastic Beanstalk runs `preinit`, `appdeploy`, and `postinit`, in this order. On subsequent deployments to running instances, Elastic Beanstalk runs `appdeploy` hooks. `configdeploy` hooks are run when a user updates instance software configuration settings. `restartappserver` hooks are run only when the user initiates an application server restart.

When your scripts encounter errors, they can exit with a non-zero status and write to `stderr` to fail the operation. The message that you write to `stderr` will appear in the event that is output when the operation fails. Elastic Beanstalk also captures this information in the log file `/var/log/eb-activity.log` If you don't want to fail the operation, return 0 (zero). Messages that you write to `stderr` or `stdout` appear in the deployment logs, but won't appear in the event stream unless the operation fails.

Packer instance cleanup

In certain circumstances, such as stopping the Packer builder process before it is finished, instances launched by Packer are not cleaned up. These instances are not part of the Elastic Beanstalk environment and can be viewed and terminated only by using the Amazon EC2 service.

To manually clean up these instances

1. Open the Amazon EC2 console.
2. Make sure you are in the same AWS Region in which you created the instance with Packer.

3. Under **Resources**, choose *N* **Running Instances**, where *N* indicates the number of running instances.
4. Click in the query text box.
5. Select the **Name** tag.
6. Enter **packer**.

   The query should look like: **tag:Name: packer**
7. Select any instances that match the query.
8. If the **Instance State** is **running**, choose **Actions**, **Instance State**, **Stop**, and then **Actions**, **Instance State**, **Terminate**.

Platform.yaml file format

The `platform.yaml` file has the following format.

```
version: "version-number"

provisioner:
   type: provisioner-type
   template: provisioner-template
   flavor: provisioner-flavor

metadata:
   maintainer: metadata-maintainer
   description: metadata-description
   operating_system_name: metadata-operating_system_name
   operating_system_version: metadata-operating_system_version
   programming_language_name: metadata-programming_language_name
   programming_language_version: metadata-programming_language_version
   framework_name: metadata-framework_name
   framework_version: metadata-framework_version

option_definitions:
   - namespace: option-def-namespace
     option_name: option-def-option_name
     description: option-def-description
     default_value: option-def-default_value

option_settings:
   - namespace: "option-setting-namespace"
```

```
      option_name: "option-setting-option_name"
      value: "option-setting-value"
```

Replace the placeholders with these values:

`version-number`

Required. The version of the YAML definition. Must be **`1.0`**.

`provisioner-type`

Required. The type of builder used to create the custom platform. Must be **`packer`**.

`provisioner-template`

Required. The JSON file containing the settings for *`provisioner-type`*.

`provisioner-flavor`

Optional. The base operating system used for the AMI. One of the following:

amazon (default)

Amazon Linux. If not specified, the latest version of Amazon Linux when the platform is created.

Amazon Linux 2 isn't a supported operating system flavor.

ubuntu1604

Ubuntu 16.04 LTS

rhel7

RHEL 7

rhel6

RHEL 6

`metadata-maintainer`

Optional. Contact information for the person who owns the platform (100 characters).

`metadata-description`

Optional. Description of the platform (2,000 characters).

metadata-operating_system_name

Optional. Name of the platform's operating system (50 characters). This value is available when filtering the output for the ListPlatformVersions API.

metadata-operating_system_version

Optional. Version of the platform's operating system (20 characters).

metadata-programming_language_name

Optional. Programming language supported by the platform (50 characters)

metadata-programming_language_version

Optional. Version of the platform's language (20 characters).

metadata-framework_name

Optional. Name of the web framework used by the platform (50 characters).

metadata-framework_version

Optional. Version of the platform's web framework (20 characters).

option-def-namespace

Optional. A namespace under `aws:elasticbeanstalk:container:custom` (100 characters).

option-def-option_name

Optional. The option's name (100 characters). You can define up to 50 custom configuration options that the platform provides to users.

option-def-description

Optional. Description of the option (1,024 characters).

option-def-default_value

Optional. Default value used when the user doesn't specify one.

The following example creates the option **NPM_START**.

```
options_definitions:
  -  namespace: "aws:elasticbeanstalk:container:custom:application"
     option_name: "NPM_START"
     description: "Default application startup command"
     default_value: "node application.js"
```

option-setting-namespace

Optional. Namespace of the option.

option-setting-option_name

Optional. Name of the option. You can specify up to 50 options provided by Elastic Beanstalk.

option-setting-value

Optional. Value used when the user doesn't specify one.

The following example creates the option **TEST**.

```
option_settings:
 - namespace: "aws:elasticbeanstalk:application:environment"
   option_name: "TEST"
   value: "This is a test"
```

Tagging custom platform versions

You can apply tags to your AWS Elastic Beanstalk custom platform versions. Tags are key-value pairs associated with AWS resources. For information about Elastic Beanstalk resource tagging, use cases, tag key and value constraints, and supported resource types, see Tagging Elastic Beanstalk application resources.

You can specify tags when you create a custom platform version. In an existing custom platform version, you can add or remove tags, and update the values of existing tags. You can add up to 50 tags to each custom platform version.

Adding tags during custom platform version creation

If you use the EB CLI to create your custom platform version, use the `--tags` option with **eb platform create** to add tags.

```
~/workspace/my-app$ eb platform create --tags mytag1=value1,mytag2=value2
```

With the AWS CLI or other API-based clients, add tags by using the `--tags` parameter on the **create-platform-version** command.

```
$ aws elasticbeanstalk create-platform-version \
      --tags Key=mytag1,Value=value1 Key=mytag2,Value=value2 \
      --platform-name my-platform --platform-version 1.0.0 --platform-definition-bundle
 S3Bucket=amzn-s3-demo-bucket,S3Key=sample.zip
```

Managing tags of an existing custom platform version

You can add, update, and delete tags in an existing Elastic Beanstalk custom platform version.

If you use the EB CLI to update your custom platform version, use **eb tags** to add, update, delete, or list tags.

For example, the following command lists the tags in a custom platform version.

```
~/workspace/my-app$ eb tags --list --resource "arn:aws:elasticbeanstalk:us-east-2:my-
account-id:platform/my-platform/1.0.0"
```

The following command updates the tag `mytag1` and deletes the tag `mytag2`.

```
~/workspace/my-app$ eb tags --update mytag1=newvalue --delete mytag2 \
      --resource "arn:aws:elasticbeanstalk:us-east-2:my-account-id:platform/my-
platform/1.0.0"
```

For a complete list of options and more examples, see `eb tags`.

With the AWS CLI or other API-based clients, use the **list-tags-for-resource** command to list the tags of a custom platform version.

```
$ aws elasticbeanstalk list-tags-for-resource --resource-arn
 "arn:aws:elasticbeanstalk:us-east-2:my-account-id:platform/my-platform/1.0.0"
```

Use the **update-tags-for-resource** command to add, update, or delete tags in a custom platform version.

```
$ aws elasticbeanstalk update-tags-for-resource \
      --tags-to-add Key=mytag1,Value=newvalue --tags-to-remove mytag2 \
      --resource-arn "arn:aws:elasticbeanstalk:us-east-2:my-account-id:platform/my-
platform/1.0.0"
```

Specify both tags to add and tags to update in the `--tags-to-add` parameter of **update-tags-for-resource**. A nonexisting tag is added, and an existing tag's value is updated.

Note

To use some of the EB CLI and AWS CLI commands with an Elastic Beanstalk custom platform version, you need the custom platform version's ARN. You can retrieve the ARN by using the following command.

```
$ aws elasticbeanstalk list-platform-versions
```

Use the `--filters` option to filter the output down to your custom platform's name.

Document history

The following table describes the important changes to the *AWS Elastic Beanstalk Developer Guide* since April 2024.

Change	Description	Date
Revised topic: What is AWS Elastic Beanstalk?	New architecture diagram and condensed introduction for clarity.	June 5, 2025
Revised topic: Setting up the EB CLI	Condensed and focused installation instructions.	June 5, 2025
Revised topic: PHP Quickstart	Added a step to update and redeploy the application and improved some instructions.	June 5, 2025
Revised topic: Getting started tutorial	Streamlined the Getting Started tutorial for clarity and quicker onboarding.	June 5, 2025
Reorganized content	Reorganized content structure to highlight essential information and improve discoverability.	June 5, 2025
New topic: Managing EC2 security groups	Elastic Beanstalk adds support to opt out environme nt from default EC2 security group.	April 30, 2025
New topic: Migrating IIS applications to Elastic Beanstalk	Elastic Beanstalk adds feature to migrate your Windows IIS applications to AWS Elastic Beanstalk.	April 18, 2025

New topic: Using Elastic Beanstalk with AWS Secrets Manager and AWS Systems Manager Parameter Store	Elastic Beanstalk adds support to reference AWS Secrets Manager secrets and AWS Systems Manager Parameter Store parameters with environment variables.	March 31, 2025
AWS managed policy updates for `AWSElasticBeanstalkManagedUpdatesCustomerRolePolicy` .	Updated permissions in AWS managed policy.	February 27, 2025
Launch configurations deprecated in favor of launch templates	Revised this topic to explain how Elastic Beanstalk environments are affected by Amazon EC2 Auto Scaling deprecation of *launch configurations* in favor of *launch templates*. We also revised other content related to launch configurations and launch templates.	January 23, 2025
New topic: Using EC2 Fast Launch with Windows platform branches	Elastic Beanstalk Windows platform releases include base AMIs with EC2 Fast Launch enabled.	January 22, 2025
New topic: Spot Instance allocation strategy	Elastic Beanstalk adds support for Spot Allocation Strategy configuration during environment creation.	January 15, 2025
`AdministratorAccess-AWSElasticBeanstalk` AWS managed policy	Updated permissions in AWS managed policy.	December 11, 2024

Recommendations for Graviton arm64 first wave environments to Archived content chapter	Moved EB Recommendations for Graviton arm64 first wave environments to Archived content chapter	October 5, 2024
Launch templates	New topic: Launch templates . This topic explains how Amazon EC2 Auto Scaling is phasing out *launch configura tions* in favor of *launch templates*.	October 1, 2024
QuickStartfor Python	New topic: QuickStart for Python	September 24, 2024
Required Amazon S3 bucket permissions for restrictive VPC endpoint policies	New topic: Required Amazon S3 bucket permissions for restrictive VPC endpoint policies	September 18, 2024
QuickStart for Java on Tomcat	New topic: QuickStart for Java JSP application running on Tomcat	September 12, 2024
QuickStart for Java	New topic: QuickStart for Java	September 12, 2024
Using the Amazon EC2 Systems Manager for Docker private repositories	Reinstated topic: Using the Amazon EC2 Systems Manager for Docker private repositories. Replaces topic *Using the AWS Secrets Manager for Docker private repositories* until Elastic Beanstalk / SSM integration is supported.	September 8, 2024
EB CLI 2.6 (retired) to Archived content chapter	Moved EB CLI 2.6 (retired) to Archived content chapter	August 15, 2024

EB API (retired) to Archived content chapter	Moved EB API (retired) to Archived content chapter	August 15, 2024
Deploying Elastic Beanstalk applications from Docker containers	Updated and reorganized: Deploying Elastic Beanstalk applications from Docker containers	August 15, 2024
Archived content	New chapter: Archived content	August 15, 2024
QuickStart for Windows ASP.NET	New topic: QuickStart for Windows ASP.NET	July 5, 2024
QuickStart for .NET Core on Windows	New topic: QuickStart for .NET Core on Windows	June 28, 2024
QuickStart for Docker	New topic: QuickStart for Docker	June 19, 2024
Preventing cross-environment Amazon S3 bucket access	New topic: Preventing cross-environment Amazon S3 bucket access	June 12, 2024
QuickStart for .NET Core on Linux	New topic: QuickStart for .NET Core on Windows	May 28, 2024
QuickStart for PHP	New topic: QuickStart for PHP	May 10, 2024
QuickStart for Node.js	New topic: QuickStart for Node.js	May 5, 2024
QuickStart for Go	New topic: QuickStart for Go	May 5, 2024

Elastic Beanstalk platform release schedule	Added new topic that includes a schedule of Upcoming platform branch releases. Moved Retiring platform branch schedule and Retired platform branch history to this topic.	May 1, 2024
`AWSElasticBeanstalkRoleCore` AWS managed policy	Updated permissions in AWS managed policy.	April 30, 2024
`AWSElasticBeanstalkManagedUpdatesServiceRolePolicy` AWS managed policy	Updated permissions in AWS managed policy.	April 30, 2024
`AWSElasticBeanstalkManagedUpdatesInternalServiceRolePolicy` AWS managed policy	Updated permissions in AWS managed policy.	April 30, 2024
`AWSElasticBeanstalkMaintenance` AWS managed policy	Updated permissions in AWS managed policy.	April 30, 2024
`AWSElasticBeanstalkInternalMaintenanceRolePolicy` AWS managed policy	Updated permissions in AWS managed policy.	April 30, 2024