AWS App2Container

User Guide

AWS App2Container: User Guide

Table of Contents

What is AWS App2Container?

AWS App2Container (A2C) is a command line tool to help you lift and shift applications that run in your on-premises data centers or on virtual machines, so that they run in containers that are managed by Amazon ECS or Amazon EKS.

Moving legacy applications to containers is often the starting point toward application modernization. There are many benefits to containerization:

- Reduces operational overhead and infrastructure costs
- Increases development and deployment agility
- Standardizes build and deployment processes across an organization

Contents

How App2Container works

You can use App2Container to generate container images for one or more applications running on Windows or Linux servers that are compatible with the Open Containers Initiative (OCI). This includes commercial off-the-shelf applications (COTs). App2Container does not need source code for the application to containerize it.

You can use App2Container directly on the application servers that are running your applications, or perform the containerization and deployment steps on a worker machine.

App2Container performs the following tasks:

- Creates an inventory list for the application server that identifies all running ASP.NET (Windows) and Java applications (Linux) that are candidates to containerize.
- Analyzes the run-time dependencies of supported applications that are running, including cooperating processes and network port dependencies.
- Extracts application artifacts for containerization and generates a Dockerfile.
- Initiates builds for the application container.
- Generates the AWS artifacts needed to deploy the containers on Amazon ECS or Amazon EKS. For example:
  - A CloudFormation template to configure required compute, network, and security infrastructure to deploy containers using Amazon ECS or Amazon EKS.
  - An Amazon ECR container image, Amazon ECS task definitions, and a Kubernetes deployment yaml that incorporate best practices for security and scalability of the application by integrating with various AWS services.
  - CI/CD pipelines for AWS CodeStar (CodeCommit, CodeBuild and CodeDeploy) to build and deploy containers.

Accessing AWS through App2Container

When you initialize App2Container, you provide it with your AWS credentials. This allows App2Container to store artifacts in Amazon S3, if you configured it to do so, and to create and deploy application containers using AWS services such as Amazon ECS and Amazon EKS.

Pricing

App2Container is offered at no additional charge. You are charged only when you use other AWS services to run your containerized application, such as Amazon ECR, Amazon ECS, and Amazon EKS. For more information, see AWS Pricing.

Applications you can containerize using AWS App2Container

App2Container supports the following application types:

- Java applications (Linux)
- ASP.NET applications (Windows)

During inventory and analysis steps, App2Container flags applications that were developed in ASP.NET or Java stacks where the framework version or specific features are not supported. This can help identify what steps you need to take to prepare applications for running in containers.

Important
App2Container does not containerize database layer components. If your application requires access to a database, you must configure your application container to have access to the database server.

Supported applications for Linux

App2Container for Linux supports identification and containerization of Java applications. The CLI identifies Java processes, and can generate container images that replicate the running state of each process.

Supported Java application frameworks

- Tomcat
- Spring Boot
- JBoss (standalone mode)
- Weblogic (standalone mode)
- Websphere (standalone mode)

Supported Linux distributions

- Ubuntu
- CentOS
- RHEL
- Amazon Linux

Unsupported for Java application frameworks

- Cluster/HA mode

Supported applications for Windows

App2Container can identify and containerize ASP.NET applications deployed on IIS, running on Windows Server 2016 or later. It supports Windows Server Core as a base image corresponding to the Windows Server version of an application server or worker node.

Application framework and system requirements

- Windows Server Core 2016 or 2019 (prior operating systems do not support containers)
- IIS 7.5 or later
- .NET framework version 3.5 or later

Unsupported applications

- ASP.NET applications that depend on WCF
- ASP.NET applications that use files and registries outside of IIS web application directories
- ASP.NET applications that depend on other Windows services or processes outside of IIS
- ASP.NET applications that depend on features of a Windows operating system version prior to Windows Server Core 2016

If your applications are running on Windows Server 2008 or 2012 R2, you might still be able to use App2Container by setting up a worker machine for containerization and deployment steps.

Setting up AWS App2Container

Complete these tasks before you use App2Container for the first time.

Tasks

Sign up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all services in AWS. You are charged only for the services that you use.

If you do not have an AWS account already, use the following procedure to create one.

To create 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 and entering a verification code on the phone keypad.

Grant permissions to IAM users

App2Container needs access to AWS services in order to run most of its commands. There are two very different sets of permissions needed to run **app2container** commands.

- The general purpose user or group can run all of the commands *except* commands that are run with the `--deploy` option.
- For deployment, App2Container must be able to create or update AWS objects for container management services (Amazon ECR with Amazon ECS or Amazon EKS), and to create pipelines with AWS CodeStar services. This requires elevated permissions that should only be used for deployment.

We recommend that you create general purpose IAM resources, and if you plan to use App2Container to deploy your containers or create pipelines, that you create separate IAM resources for deployment.

For more information and instructions about setting up IAM resources for App2Container, see Identity and access management in App2Container (p. 38).

Decide where containerization will run

To use App2Container on the server where the applications are running, you must set up an AWS profile, install App2Container, and install the Docker engine. If your server does not meet the requirements to containerize your application and deploy it to AWS, or if you do not want to install the Docker engine on the application server, you can set up and use a worker machine. On the worker machine, you do the steps to containerize your application and deploy it to AWS. The following are example situations where you might decide to set up a worker machine:

- Your application servers are running in an on-premises data center and they do not have internet access.
- Your application server is running on a Windows operating system that does not support containers. For more information, see Supported applications (p. 3).
- You prefer to use a dedicated server to run the containerization and deployment steps.

When you set up a worker machine to handle the steps to containerize and deploy your applications, it must have the same base operating system as your application server (Linux or Windows), and the operating system must support containers. We recommend that you launch an Amazon EC2 instance as the worker machine, using an Amazon Machine Image (AMI) that is optimized for Amazon ECS.

Learn more

- How App2Container works (p. 1)
- Amazon ECS-optimized AMIs in the *Amazon Elastic Container Service Developer Guide*
- Launching an instance using the Launch Instance Wizard in the *Amazon EC2 User Guide for Linux Instances*
- Launching an instance using the Launch Instance Wizard in the *Amazon EC2 User Guide for Windows Instances*

Configure the AWS profile

AWS App2Container requires access to AWS resources for containerization and deployment commands. It uses information from your AWS profile to configure access to AWS resources for your account.

To configure the AWS profile information, install the AWS Command Line Interface (AWS CLI) or AWS Tools for Windows PowerShell on the application server and worker machine (if using). After you containerize your applications, you can also use the AWS CLI or Tools for Windows PowerShell to deploy them on AWS, though we recommend using the `--deploy` option with the **generate app-deployment** and **generate pipeline** commands to do your deployment.

Options

1. Install the AWS CLI — For more information, see Installing the AWS CLI and Configuration basics in the *AWS Command Line Interface User Guide*.
2. Install the Tools for Windows PowerShell — For more information, see Installing the AWS Tools for Windows PowerShell and Shared credentials in the *AWS Tools for Windows PowerShell User Guide*.

   Note

   - Tools for Windows PowerShell is required for running App2Container commands in PowerShell on a Windows server.
   - Tools for Windows PowerShell comes pre-installed on Windows-based Amazon Machine Images (AMIs). If your application server or worker machine is an Amazon EC2 instance that was launched from one of these AMIs, you can skip to configuring your AWS profile. See Shared credentials in the *AWS Tools for Windows PowerShell User Guide* for more details.

Install the Docker engine

App2Container uses the Docker engine (Docker CE) to create container images and generate Dockerfiles that run the containers hosted on Amazon ECS or Amazon EKS. You must install the Docker engine on the application server or worker machine that you'll use to containerize the application using the **containerize** command.

Install Docker on Linux

Use the following procedure to install Docker on Linux.

To install the Docker engine

1. **Install Docker**

   Choose your Linux distribution from the following options, and follow instructions to download and install the Docker engine, using the links provided.

   Amazon Linux

   To download and install the Docker engine on Amazon Linux instances, see Docker basics for Amazon ECS in the *Amazon Elastic Container Service Developer Guide*. This works with any Amazon Linux instance.

   RHEL

   Recent versions of RHEL do not natively support the Docker engine. However, you can still download and install the Docker engine on RHEL to create containers that will be hosted and run on Amazon ECS or Amazon EKS. To do this, follow the instructions given for CentOS on the Docker website: Install Docker engine.

   All other supported distributions (CentOS, Ubuntu)

   To download and install the Docker engine for other supported Linux distributions, follow the instructions for your Linux distribution on the Docker website: Install Docker engine.

2. **Verify the Docker installation**

   To verify that your Docker installation was successful, run the following command.

   ```
   $ docker run -it hello-world
   ```

   When the command runs, it pulls the latest hello-world application from the Docker repository, if applicable. When the application has finished downloading, it displays a "Hello" message followed by information on how this command verified your installation of Docker.

Install Docker on Windows

Use the following procedure to install Docker on Windows.

To install the Docker engine

1. **Install Docker**

   To download and install the Docker engine on Windows, see Get started: Prep Windows for containers (Install Docker section).

2. **Verify the Docker installation**

   To verify that your Docker installation was successful, run the following command.

   ```
   PS> docker run -it hello-world
   ```

When the command runs, it pulls the latest hello-world application from the Docker repository, if applicable. When the application has finished downloading, it displays a "Hello" message followed by information on how this command verified your installation of Docker.

Getting started with AWS App2Container

AWS App2Container is a tool that helps you break down the work of moving your applications into containers, and configuring them to be hosted in AWS using the Amazon ECS or Amazon EKS container management services. Explore the resources listed below to help you get started with containers. Or to get started using App2Container commands, skip to the tutorial for the operating system that your application runs on.

Understanding Docker containers

The following resources can help you get the most out of your application containers by understanding what goes into them.

- To learn more about Docker containers on AWS, see What is Docker?.
- Use the Docker command line reference to look up Docker commands. See Use the Docker command line.

Tutorials

These tutorials walk you through the basics of using App2Container to containerize your applications.

- Containerizing a Java application on Linux (p. 9)
- Containerizing a .NET application on Windows (p. 13)

Containerizing a Java application on Linux

This tutorial takes you through the steps to containerize a legacy Java application on Linux using App2Container, and to deploy it on Amazon ECS or Amazon EKS. You can complete all steps on the application server, or you can perform the initial steps on the application server and perform the containerization and deployment steps on a worker machine.

Tasks

Prerequisites

Verify that you have completed the following prerequisites:

- You installed the AWS CLI and configured the AWS profile on your server. See Configure the AWS profile (p. 6) in the Setting up section of this user guide for more information.
- You installed the Docker engine on the server where you are running containerization and deployment steps. See Install the Docker engine (p. 6) in the Setting up section of this user guide for more information.
- There are one or more Java applications running on the application server.
- You have root access on the application server (and worker machine, if using).
- The application server (and worker machine, if using) has **tar** and 20 GB of free space.

Step 1: Install App2Container

App2Container for Linux is packaged as a tar.gz archive. The archive contains an interactive shell script that installs App2Container on your server. If you are using an application server and a worker machine, you must install App2Container on both.

To download and install App2Container for Linux

1. Download the installation file in one of the following ways:

   - Use the **curl** command to download the App2Container installation package from Amazon S3.

     ```
     $ curl -o AWSApp2Container-installer-linux.tar.gz https://app2container-release-
     us-east-1.s3.us-east-1.amazonaws.com/latest/linux/AWSApp2Container-installer-
     linux.tar.gz
     ```

   - Use your browser to download the installer from the following URL: https://app2container-release-us-east-1.s3.us-east-1.amazonaws.com/latest/linux/AWSApp2Container-installer-linux.tar.gz.

2. Extract the package to a local folder on the server.

   ```
   $ sudo tar xvf AWSApp2Container-installer-linux.tar.gz
   ```

3. Run the install script that you extracted from the package and follow the prompts.

   ```
   $ sudo ./install.sh
   ```

You can check the downloaded tar.gz installer archive for integrity by validating the MD5 and SHA256 hashes of the local file against the published hash files.

To verify the authenticity of the download

1. **Generate hashes to verify**

   From the directory where you downloaded your tar.gz installer, run the following commands to generate the hash of the downloaded tar.gz file.

   ```
   $ md5sum AWSApp2Container-installer-linux.tar.gz
   a0a1234f567bf89012345a6ce7bf89a0 AWSAppContainerization-installer-linux.tar.gz
   $ sha256sum AWSApp2Container-installer-linux.tar.gz
   ```

```
01bc2d345f6789012345bd6aa789012345d67dea8b9c0f1234ee5a67890123d4
 AWSAppContainerization-installer-linux.tar.gz
```

2. **Verify hashes against public files**

   Download the App2Container hash files from Amazon S3 using the following links, and compare the contents to the hashes that you generated in step 1:

   - AWSApp2Container-installer-linux.tar.gz.md5.
   - AWSApp2Container-installer-linux.tar.gz.sha256.

Step 2: Initialize App2Container

On each server where you installed App2Container, run the init (p. 58) command as follows.

```
$ sudo app2container init
```

You are prompted to provide the following information. Choose *<enter>* to accept the default value.

- Workspace directory path - A local directory where App2Container can store artifacts during the containerization process. The default is `/root/app2container`.
- AWS profile - Contains information needed to run App2Container, such as your AWS access keys. For more information, see Configure the AWS profile (p. 6). Your AWS defalt profile is used if you don't specify another value.
- Amazon S3 bucket - You can optionally provide the name of an Amazon S3 bucket where you can extract artifacts using the **extract** command. The **containerize** command uses the extracted components to create the application container if the Amazon S3 bucket is configured. The default is no bucket.
- Permission to collect usage metrics - You can optionally allow App2Container to collect information about the host operating system, application type, and the **app2container** commands that you run. The default is to allow the collection of metrics.
- Whether to enforce signed images - You can optionally require that images are signed using Docker Content Trust (DCT). The default is no.

Step 3: Analyze your application

On the application server, use the following procedure to prepare to containerize the application.

To prepare for containerization

1. Run the inventory (p. 60) command as follows to list the Java applications that are running on your server.

   ```
   $ sudo app2container inventory
   ```

   The output includes a JSON object collection with one entry for each application. Each application object will include key/value pairs as shown in the following example.

   ```
   "java-app-id": {
       "processId": pid,
       "cmdline": "/user/bin/java ...",
       "applicationType": "java-apptype"
   ```

```
}
```

2. Locate the application ID for the application to convert in the JSON output of the **inventory** command, and then run the analyze (p. 46) command as follows, replacing *java-app-id* with the application ID that you located.

   ```
   $ sudo app2container analyze --application-id java-app-id
   ```

   The output is a JSON file, `analysis.json`, stored in the workspace directory that you specified when you ran the **init** command.
3. (Optional) You can edit the information in the `containerParameters` section of `analysis.json` as needed before continuing to the next step.

Step 4: Transform your application

The transform phase depends on whether you are running all steps on the application server, or are using the application server for the analysis and a worker machine for containerization and deployment.

To containerize the application on the application server

If you are using an application server for all steps, run the containerize (p. 48) command as follows.

```
$ sudo app2container containerize --application-id java-app-id
```

The output is a set of deployment files that are stored in the workspace directory that you specified when you ran the **init** command.

To containerize the application on a worker machine

If you are using a worker machine for containerization and deployment, use the following procedure to transform the application.

1. On the application server, run the extract (p. 50) command as follows.

   ```
   $ sudo app2container extract --application-id java-app-id
   ```
2. If you specified an Amazon S3 bucket when you ran the **init** command, the archive is extracted to that location. Otherwise, you can manually copy the resulting archive file to the worker machine.
3. On the worker machine, run the containerize (p. 48) command as follows.

   ```
   $ sudo app2container containerize --input-archive /path/extraction-file.tar
   ```

   The output is a set of deployment artifacts that are stored in the workspace directory that you specified when you ran the **init** command.

Step 5: Deploy your application

Review the deployment artifacts generated in the previous step and modify them as needed. Run the generate app-deploy (p. 51) command as follows to deploy the application on AWS.

```
$ sudo app2container generate app-deployment --deploy --application-id java-app-id
```

Step 6: Clean up

To remove App2Container from your application server or worker machine, delete the `/usr/local/app2container` folder where it is installed, and then remove this folder from your path.

Containerizing a .NET application on Windows

This tutorial takes you through the steps to containerize a legacy .NET application running in IIS on Windows using App2Container, and to deploy it on Amazon ECS or Amazon EKS. You can complete all steps on the application server, or you can perform the initial steps on the application server and perform the containerization and deployment steps on a worker machine.

Tasks

Prerequisites

Verify that you have completed the following prerequisites:

- You installed the AWS Tools for Windows PowerShell to configure the AWS profile on your server. See Configure the AWS profile (p. 6) in the Setting up section of this user guide for more information.
- You installed the Docker engine on the server where you are running containerization and deployment steps. See Install the Docker engine (p. 6) in the Setting up section of this user guide for more information.
- There are one or more applications running in IIS on the application server.
- You are a Windows administrator on the application server (and worker machine, if using).
- The application server or worker machine has PowerShell version 5.1 or later and at least 20-30 GB of free space.

Step 1: Install App2Container

App2Container for Windows is packaged as a zip archive. The package contains a PowerShell script that installs App2Container. If you are using an application server and a worker machine, you must install App2Container on both.

To download and install App2Container for Windows

1. Download the App2Container installation package, AWSApp2Container-installer-windows.zip.
2. Extract the package to a local folder on the server and navigate to that folder.
3. Run the install script from the folder where you extracted it, and follow the prompts.

```
PS> .\install.ps1
```

4. (Optional) To verify the authenticity of the download, use the `Get-AuthenticodeSignature` PowerShell command as follows to get the Authenticode Signature of the App2Container executable.

```
PS> Get-AuthenticodeSignature C:\Users\Administrator\app2container\AWSApp2Container\bin
\app2container.exe
```

Step 2: Initialize App2Container

On each server where you installed App2Container, run the init (p. 58) command as follows.

```
PS> app2container init
```

You are prompted to provide the following information. Choose *<enter>* to accept the default value.

- Workspace directory path - A local directory where App2Container can store artifacts during the containerization process. The default is `C:\Users\Administrator\AppData\Local\app2container`.
- AWS profile - Contains information needed to run App2Container, such as your AWS access keys. For more information, see Configure the AWS profile (p. 6). Your AWS defalt profile is used if you don't specify another value.
- Amazon S3 bucket - You can optionally provide the name of an Amazon S3 bucket where you can extract artifacts using the **extract** command. The **containerize** command uses the extracted components to create the application container if the Amazon S3 bucket is configured. The default is no bucket.
- Permission to collect usage metrics - You can optionally allow App2Container to collect information about the host operating system, application type, and the **app2container** commands that you run. The default is to allow the collection of metrics.
- Whether to enforce signed images - You can optionally require that images are signed using Docker Content Trust (DCT). The default is no.

Step 3: Analyze your application

On the application server, use the following procedure to prepare to containerize the application.

To prepare for containerization

1. Run the inventory (p. 60) command as follows to list the ASP.NET applications that are running on your server.

```
PS> app2container inventory
```

The output includes a JSON object collection with one entry for each application. Each application object will include key/value pairs as shown in the following example.

```
"iis-app-id": {
    "siteName": My site name,
    "bindings": "http/*:80:",
    "applicationType": "iis",
    "discoveredWebApps": [
        "app1",
        "app2"
    ]
}
```

2. Locate the application ID for the application to convert in the JSON output of the **inventory** command, and then run the analyze (p. 46) command as follows, replacing *iis-app-id* with the application ID that you located.

   ```
   PS> app2container analyze --application-id iis-app-id
   ```

   The output is a JSON file, `analysis.json`, stored in the workspace directory that you specified when you ran the **init** command.
3. (Optional) You can edit the information in the `containerParameters` section of `analysis.json` as needed before continuing to the next step.

Step 4: Transform your application

The transform phase depends on whether you are running all steps on the application server or using the application server for the analysis and a worker machine for containerization and deployment.

To containerize the application on the application server

If you are using an application server for all steps, run the containerize (p. 48) command as follows.

```
PS> app2container containerize --application-id iis-app-id
```

The output is a set of deployment files stored in the workspace directory that you specified when you ran the **init** command.

To containerize the application on a worker machine

If you are using a worker machine for containerization and deployment, use the following procedure to transform the application.

1. On the application server, run the extract (p. 50) command as follows.

   ```
   PS> app2container extract --application-id iis-app-id
   ```

2. If you specified an Amazon S3 bucket when you ran the **init** command, the archive is extracted to that location. Otherwise, you can manually copy the resulting archive file to the worker machine.
3. On the worker machine, run the containerize (p. 48) command as follows.

   ```
   PS> app2container containerize --input-archive drive:\path\extraction-file.zip
   ```

   The output is a set of deployment artifacts that are stored in the workspace directory that you specified when you ran the **init** command.

Step 5: Deploy your application

Review the deployment artifacts generated in the previous step and modify them as needed. Run the generate app-deploy (p. 51) command as follows to deploy the application on AWS.

```
PS> app2container generate app-deployment --deploy --application-id iis-app-id
```

Applications using Windows authentication

For applications using Windows authentication, you can use the `gMSAParameters` inside of the `deployment.json` file to set the gMSA-related artifacts automatically during generation of your AWS CloudFormation template.

Perform the actions in the list below once per Active Directory domain before you update the gMSA parameters.

- Set up a secret in SecretsManager that stores the Domain credentials with the following key value pairs:

Key	Value
Username	<DomainNetBIOSName>\<DomainUser>
Password	<DomainUserPassword>

- For the VPC with the Domain Controller, verify that the DHCP options are set to reach the Domain Controller. The options for `DomainName` and `DomainNameServers` must be set correctly. See DHCP options sets for more information about how to set DHCP options.

Step 6: Clean up

To remove App2Container from your application server or worker machine, delete the `C:\Users\Administrator\app2container` folder where it is installed, and then remove this folder from your path.

Configuring your application

Containerizing your application and creating pipelines with App2Container requires configuration throughout the process. This section of the guide describes the files that are created by **app2container** commands, the fields that they contain, and which fields are configurable. App2Container commands primarily generate JSON files, using standard JSON notation. Field details for the files included here indicate where there are specific requirements for the values.

Creating IAM resources is covered separately, under the Security section. For more information and instructions about how to set up IAM resources for App2Container, see Identity and access management in App2Container (p. 38).

Contents

Configuring application containers

This topic contains information about the files that are used for configuring application containers.

Container configuration files

- analysis.json file (p. 17)

analysis.json file

When you run the **analyze (p. 46)** command, an `analysis.json` file is created for the application that is specified in the `--application-id` parameter. The **containerize** command uses this file to build the application container image and to generate artifacts.

You can configure the fields in the `containerParameters` section before running the **containerize** command to customize your application container. For configurable key/value pairs that do not apply to your container, set string values to an empty string, numeric values to zero, and Boolean values to false.

Choose one of the following options for application language and host operating system to see more detailed file information.

Java application analysis file

The Java application `analysis.json` file includes the following content:

Read-only data

- **Control fields** – Fields having to do with file creation, such as template version, and the file creation timestamp.
- **analysisInfo** – Identifies system dependencies for the application.

Configurable data

The `containerParameters` section contains the following fields:

- **imageRepository** (string, required) – The name of the repository where the application container image is stored.
- **imageTag** (string, required) – a tag for the build version of the application container image.
- **containerBaseImage** (string, required) – the base operating system image for the container build. *Must be an image name from your registry in the format <image name>[:<tag>]. Tag is optional if the repository supports "latest".*
- **appExcludedFiles** (array of strings) – specific files and directories to exclude from the container build.
- **appSpecificFiles** (array of strings) – specific files and directories to include in the container build.
- **applicationMode** (Boolean, required) – switch to an optimized mode of generating container images (supported only for standalone Java JBoss and Java Tomcat applications).
- **logLocations** (array of strings) – specific log files or log directories to be routed to `stdout`. This enables applications that write to log files on the host to be integrated with AWS services such as CloudWatch and Kinesis Data Firehose.
- **enableDynamicLogging** (Boolean, required) – maps application logs to `stdout` as they are created. *If set to true, requires log directories to be entered in `logLocations`.*
- **dependencies** (array of strings) – a listing of all dependent processes or applications found for the application ID by the **analyze** command. You can remove specific dependencies to exclude them from the container.

The following example shows an `analysis.json` file for a Java application running on Linux.

```
{
 "a2CTemplateVersion": "",
       "createdTime": "",
       "containerParameters": {
              "_comment1": "*** EDITABLE: The below section can be edited according to the
 application requirements. Please see the analysisInfo section below for details discovered
 regarding the application. ***",
              "imageRepository": "java-tomcat-6e6f3a87",
              "imageTag": "latest",
              "containerBaseImage": "ubuntu:18.04",
              "appExcludedFiles": [],
              "appSpecificFiles": [],
              "applicationMode": true,
              "logLocations": [],
              "enableDynamicLogging": false,
              "dependencies": []
       },
       "analysisInfo": {
              "_comment2": "*** NON-EDITABLE: Analysis Results ***",
              "processId": 2065,
              "appId": "java-tomcat-6e6f3a87",
              "userId": "1000",
              "groupId": "1000",
              "cmdline": [
                     "/usr/bin/java",
                     "... list of commands",
                     "start"
              ],
              "osData": {
                     "BUG_REPORT_URL": "",
                     "HOME_URL": "",
                     "ID": "ubuntu",
                     "ID_LIKE": "debian",
                     "NAME": "Ubuntu",
                     "PRETTY_NAME": "Ubuntu 18.04.2 LTS",
                     "PRIVACY_POLICY_URL": "",
                     "SUPPORT_URL": "",
                     "UBUNTU_CODENAME": "",
                        "VERSION": "",
                        "VERSION_CODENAME": "",
                        "VERSION_ID": "18.04"
                },
                "osName": "ubuntu",
                "ports": [
                        {
                                "localPort": 8080,
                                "protocol": "tcp6"
                        },
                        {
                                "localPort": 8009,
                                "protocol": "tcp6"
                        },
                        {
                                "localPort": 8005,
                                "protocol": "tcp6"
                        }
                ],
                "Properties": {
                        "catalina.base": "<application directory>",
                        "catalina.home": "<application directory>",
                        "classpath": "<application directory>/bin/bootstrap.jar:... etc.",
                        "ignore.endorsed.dirs": "",
                        "java.io.tmpdir": "<application directory>/temp",
                        "java.protocol.handler.pkgs": "org.apache.catalina.webresources",
                        "java.util.logging.config.file": "<application directory>/conf/
logging.properties",
                        "java.util.logging.manager": "org.apache.juli.ClassLoaderLogManager",
                        "jdk.tls.ephemeralDHKeySize": "2048",
                        "jdkVersion": "11.0.7",
                        "org.apache.catalina.security.SecurityListener.UMASK": ""
                },
                "AdvancedAppInfo": {
                        "Directories": {
                                "base": "<application directory>",
                                "bin": "<application directory>/bin",
                                "conf": "<application directory>/conf",
                                "home": "<application directory>",
                                "lib": "<application directory>/lib",
                                "logConfig": "<application directory>/conf/logging.properties",
                                "logs": "<application directory>/logs",
                                "tempDir": "<application directory>/temp",
                                "webapps": "<application directory>/webapps",
                                "work": "<application directory>/work"
                        },
                        "distro": "java-tomee",
                        "flavor": "plume",
                        "jdkVersion": "11.0.7",
                        "version": "8.0.0"
                },
                "env": {
                        "HOME": "... Java Home directory",
                        "JDK_JAVA_OPTIONS": "",
                        "LANG": "C.UTF-8",
                        "LC_TERMINAL": "iTerm2",
                        "LC_TERMINAL_VERSION": "3.3.11",
                        "LESSCLOSE": "/usr/bin/lesspipe %s %s",
                        "LESSOPEN": "| /usr/bin/lesspipe %s",
                        "LOGNAME": "ubuntu",
                        "LS_COLORS": "",
                        "MAIL": "",
                        "OLDPWD": "",
                        "PATH": "... server PATH",
                        "PWD": "",
                        "SHELL": "/bin/bash",
                        "SHLVL": "1",
                        "SSH_CLIENT": "",
                        "SSH_CONNECTION": "",
                        "SSH_TTY": "",
                        "TERM": "",
                        "USER": "ubuntu",
                        "XDG_DATA_DIRS": "",
                        "XDG_RUNTIME_DIR": "",
                        "XDG_SESSION_ID": "1",
                        "_": "bin/startup.sh"
                 },
                 "cwd": "",
                 "procUID": {
                        "euid": "1000",
                        "suid": "1000",
                        "fsuid": "1000",
                        "ruid": "1000"
                 },
                 "procGID": {
                        "egid": "1000",
                        "sgid": "1000",
                        "fsgid": "1000",
                        "rgid": "1000"
                 },
                 "userNames": {
                        "1000": "ubuntu"
                 },
                 "groupNames": {
                        "1000": "ubuntu"
                 },
                 "fileDescriptors": [
                        "<application directory>/logs/... log files",
                        "<application directory>/lib/... jar files",
                        "... etc.",
                        "/usr/lib/jvm/.../lib/modules"
                 ],
                 "dependencies": {}
             }
   }
```

.NET application analysis file

The .NET application `analysis.json` file includes the following content:

Read-only data

- **Control fields** – Fields having to do with file creation, such as template version, and the file creation timestamp.
- **analysisInfo** – Identifies system dependencies for the application.

Configurable data

The `containerParameters` section contains the following fields:

- **containerBaseImage** (string, required) – the base operating system image for the container build. *Must be an image name from your registry in the format <image name>[:<tag>]. Tag is optional if the repository supports "latest".*
- **enableServerConfigurationUpdates** (Boolean, required) – provides an option in the Dockerfile to restore the application configuration of the source server.
- **imageRepositoryName** (string, required) – The name of the repository where the application container image is stored.

- **imageTag** (string, required) – a tag for the build version of the application container image.
- **includedWebApps** (array of strings) – the application IDs for web applications running under the IIS site that should be included in the container image. *Applications must have been running in IIS during inventory and analysis.*
- **additionalExposedPorts** (array of numbers) – additional port numbers that should be exposed inside of the application container.
- **appIncludedFiles** (array of strings) – specific files and directories to include in the container build.
- **enableLogging** (Boolean, required) – enables dynamic logging, redirecting application logs to container `stdout`.

The following example shows an `analysis.json` file for a .NET application running on Windows.

```
{
  "a2CTemplateVersion": "3.1",
  "createdTime": "",
  "containerParameters": {
    "_comment": "*** EDITABLE: The below section can be edited according to the application
 requirements. Please see the Analysis Results section further below for details discovered
 regarding the application. ***",
    "containerBaseImage": "mcr.microsoft.com/dotnet/framework/aspnet:4.7.2-
windowsservercore-ltsc2019",
    "enableServerConfigurationUpdates": true,
    "imageRepositoryName": "iis-smarts-51d2dbf8",
    "imageTag": "latest",
    "includedWebApps": [

    ],
    "additionalExposedPorts": [

    ],
    "appIncludedFiles": [

    ],
    "enableLogging": false
  },
  "analysisInfo": {
    "_comment": "*** NON-EDITABLE: Analysis Results ***",
    "hostInfo": {
      "os": "...",
      "osVersion": "...",
      "osWindowsDirectory": "...",
      "arch": "..."
    },
    "appId": "iis-smarts-51d2dbf8",
    "appServerIp": "localhost",
    "appType": "IIS",
    "appName": "smarts",
    "appPoolName": "smarts",
    "appPoolIdentity": "ApplicationPoolIdentity",
    "appPoolDotnetVersion": "v4.0",
    "iisVersion": "IIS 8.5",
    "ports": [
      {
        "localPort": 90,
        "protocol": "http"
      }
    ],
    "sitePhysicalPath": "<IIS web root directory>\\smarts",
    "discoveredWebApps": [

    ],
    "features": [
      "File-Services",
      "FS-FileServer",
      "Web-Http-Tracing",
      "Web-Basic-Auth",
      "Web-Digest-Auth",
      "Web-Url-Auth",
      "Web-Windows-Auth",
      "Web-ASP",
      "Web-CGI",
      "Web-Mgmt-Tools",
      "Web-Mgmt-Console",
      "Web-Scripting-Tools",
      "FS-SMB1",
      "User-Interfaces-Infra",
      "Server-Gui-Mgmt-Infra",
      "Server-Gui-Shell",
      "PowerShell-ISE"
    ],
    "siteUsesWindowsAuth": true,
    "serverBackupFile": "<application directory>\\Web Deploy Backups\\... backup zip file",
    "reportPath": "<application output directory>\\iis-smarts-51d2dbf8\\report.txt"
  }
}
```

Configuring container deployment

This topic contains information about the files that are used for configuring deployment of application containers.

Container deployment files

- deployment.json file (p. 22)

deployment.json file

When you run the **containerize (p. 48)** command, a `deployment.json` file is created for the application specified in the `--application-id` parameter. The **generate app-deployment** command uses this file, along with others, to generate application deployment artifacts. All of the fields in this file are configurable as needed to customize your application container deployment before running the **generate app-deployment** command.

> **Important**
> The `deployment.json` file includes sections for both Amazon ECS and Amazon EKS. You must set the Boolean value deployment flag for the section that matches your target container management service to true and the other flag to false. The flag to deploy to Amazon ECS is `createEcsArtifacts`, and the flag to deploy to Amazon EKS is `createEksArtifacts`.

The application `deployment.json` file includes the following content. While all fields are configurable, the following fields should not be changed: `a2CTemplateVersion`, `applicationId`, and `imageName`. For key/value pairs that do not apply to your deployment, set string values to an empty string, numeric values to zero, and Boolean values to false.

- **exposedPorts** (array of objects, required) – an array of JSON objects representing the ports that should be exposed when the container is running. Each object consists of the following fields:
  - **localPort** (number) – a port to expose for container communication.
  - **protocol** (string) – the application protocol for the exposed port. For example, "http".
- **environment** (array of objects) – environment variables to be passed on to the target container management deployment. For Amazon ECS deployments, the key/value pairs update the ECS

task definition. For Amazon EKS deployments, the key/value pairs update the Kubernetes `deployment.yml` file.

- **ecrParameters** (object) – contains parameters needed to register application container images in Amazon ECR.
  - **ecrRepoTag** (string, required) – the version tag to use for registering an application container image in Amazon ECR.
- **ecsParameters** (object) – contains parameters needed for deployment to Amazon ECS. *The createEcsArtifacts parameter is always required. Other parameters in this section that are marked as required apply only to ECS deployment.*
  - **createEcsArtifacts** (Boolean, required) – a flag that indicates if you are targeting Amazon ECS for deployment.
  - **ecsFamily** (string, required) – an ID for the ECS family in the ECS task definition. We recommend setting this value to the application ID.
  - **cpu** (number or string, required) – the hard limit of CPU units to present for the task. It can be expressed as an integer in the ECS task definition, using CPU units, for example 1024, or as a string using vCPUs, for example 1 vCPU or 1 vcpu. When the task definition is registered, a vCPU value is converted to an integer indicating the CPU units.
  - **memory** (number or string, required*) – the hard limit of memory (in MiB) to present to the task. It can be expressed as an integer in the ECS task definition, using MiB, for example 1024, or as a string using GB, for example 1GB. When the task definition is registered, a GB value is converted to an integer indicating the MiB.

    ** This parameter is required for Linux containers, but is not supported for Windows containers.*
  - **dockerSecurityOption** (string) – for .NET applications, this is the gMSA Credspec location value for the ECS task definition.
  - **enableCloudwatchLogging** (Boolean, required*) – a flag that sets the ECS task definition to enable CloudWatch logging for your Windows application container. *If set to true, the `enableDynamicLogging` field in the `analysis.json` file must have a valid value.*

    ** This parameter is required for Windows containers, but is not supported for Linux containers.*
  - **publicApp** (Boolean, required) – a flag to configure the CloudFormation templates with a public endpoint for your application when it runs.
  - **stackName** (string, required) – a name to use as a prefix to your CloudFormation stack application resource name (arn). We recommend using the application ID for this.
  - **reuseResources** (object) – contains shared resource identifiers that can be used throughout your CloudFormation templates.
    - **vpcId** (string) – the VPC ID, if you want to bring your own VPC or to reuse an existing VPC that App2Container created for a prior deployment.
    - **cfnStackName** (string) – the name or ID (arn) of the CloudFormation stack created with App2Container for the containerized application.
    - **sshKeyPairName** (string) – the name of the Amazon EC2 key pair to use for the instances that your container runs on.
  - **gMSAParameters** (object) – contains parameters used by the CloudFormation template to create gMSA-related artifacts for .NET applications.
    - **domainSecretsArn** (string) – the Secrets Manager arn for the domain credentials to join the ECS nodes to gMSA Active Directory.
    - **domainDNSName** (string) – the DNS Name of the gMSA Active Directory for ECS nodes to join.
    - **domainNetBIOSName** (string) – the Net BIOS name of the Active Directory for ECS nodes to join.
    - **createGMSA** (Boolean, required) – a flag to create a gMSA Active Directory security group and account using the name supplied in the `gMSAName` field.
    - **gMSAName** (string) – the name of the gMSA Active Directory that the container should use for access.

  - **deployTarget** (string, required) – identifies which ECS container launch type runs the task or CloudFormation template. *Valid values are "EC2" and "FARGATE".*
- **fireLensParameters** (object) – contains parameters needed to use FireLens to route your application logs for your Amazon ECS tasks. *The enableFireLensLogging parameter is always required. Other parameters in this section that are marked as required apply only when FireLens is used for log routing.*
  - **enableFireLensLogging** (Boolean, required) – a flag for using FireLens for Amazon ECS to configure application log routing for containers.
  - **logDestinations** (array of objects) – a list of unique target destinations for application log routing. If more than one destination is configured, App2Container creates a custom file that contains the FireLens configuration. Otherwise, the destination parameters are used directly in the ECS task definition and CloudFormation templates.
    - **service** (string) – the AWS service to route logs to. *Valid values are "cloudwatch", "firehose", and "kinesis".*
    - **regexFilter** (string) – A Ruby regular expression to match against log content to determine where to route the log.
    - **streamName** (string) – the name of the log delivery stream that will be created at the destination.
- **eksParameters** (object) – contains parameters needed for deployment to Amazon EKS. *The createEksArtifacts parameter is always required. Other parameters in this section that are marked as required apply only to EKS deployment.*
  - **createEksArtifacts** (Boolean, required) – a flag that indicates if you are targeting Amazon EKS for deployment.
  - **stackName** (string, required) – a name to use as a prefix to your CloudFormation stack arn. We recommend using the application ID for this.
  - **reuseResources** (object) – contains shared resource identifiers that can be used throughout your CloudFormation templates.
    - **vpcId** (string) – the VPC ID, if you want to bring your own VPC or to reuse an existing VPC that App2Container created for a prior deployment. *If you are bringing a custom VPC, you must have two or more private subnets in two or more Availability Zones, and can optionally have two or more public subnets in the same two Availability Zones.*
    - **cfnStackName** (string) – the name or ID (arn) of the CloudFormation stack created with App2Container for the containerized application.
    - **sshKeyPairName** (string) – the name of the Amazon EC2 key pair to use for the instances that your container runs on.

The following example shows a `deployment.json` file for a Java application running on Linux.

```
{
	"a2CTemplateVersion": "3.1",
	"applicationId": "java-tomcat-6e6f3a87",
	"imageName": "java-tomcat-6e6f3a87",
	"exposedPorts": [
		{
			"localPort": 8080,
			"protocol": "tcp6"
		},
		{
			"localPort": 8009,
			"protocol": "tcp6"
		},
		{
			"localPort": 8005,
			"protocol": "tcp6"
		}
	],
	"environment": [],
        "ecrParameters": {
                "ecrRepoTag": "latest"
        },
        "ecsParameters": {
                "createEcsArtifacts": true,
                "ecsFamily": "java-tomcat-6e6f3a87",
                "cpu": 2,
                "memory": 4096,
                "dockerSecurityOption": "",
                "enableCloudwatchLogging": false,
                "publicApp": true,
                "stackName": "app2container-java-tomcat-6e6f3a87-ECS",
                "reuseResources": {
                        "vpcId": "",
                        "cfnStackName": "",
                        "sshKeyPairName": ""
                },
                "gMSAParameters": {
                        "domainSecretsArn": "",
                        "domainDNSName": "",
                        "domainNetBIOSName": "",
                        "createGMSA": false,
                        "gMSAName": ""
                },
                "deployTarget": "FARGATE"
        },
        "fireLensParameters": {
               "enableFireLensLogging": true,
               "logDestinations": [
                       {
                                "service": "cloudwatch",
                                "matchRegex": "^.*INFO.*$",
                                "streamName": "Info"
                       },
                       {
                                "service": "cloudwatch",
                                "matchRegex": "^.*WARN.*$",
                                "streamName": "Warn"
                       }
               ]
        },
        "eksParameters": {
                "createEksArtifacts": false,
                "applicationName": "",
                "stackName": "java-tomcat-6e6f3a87",
                "reuseResources": {
                        "vpcId": "",
                        "cfnStackName": "",
                        "sshKeyPairName": ""
                }
        }
}
```

Configuring container pipelines

This topic contains information about the files that are used to configure and deploy application container pipelines using AWS CodeStar services.

Pipeline configuration files

- pipeline.json file (p. 26)

pipeline.json file

When you run the **generate app-deployment (p. 51)** command, a `pipeline.json` file is created for the application specified in the `--application-id` parameter. The **generate pipeline** command uses this file, along with others, to generate pipeline deployment artifacts. All of the fields in this file are configurable as needed to customize your application container pipeline before running the **generate pipeline** command .

The application `pipeline.json` file includes the following content. While all fields are configurable, the `a2CTemplateVersion` field should not be changed. For key/value pairs that do not apply to your pipeline, set string values to an empty string, numeric values to zero, and Boolean values to false.

- **sourceInfo** (object) – contains JSON objects for AWS CodeStar configuration.
  - **CodeCommit** (object) – contains parameters needed for AWS CodeCommit configuration.
    - **repositoryName** (string, required) – the name of the AWS CodeCommit repository to use or create.
    - **branch** (string, required) – the name of the code branch in the AWS CodeCommit repository to commit to.
- **imageInfo** (object) – contains parameters needed for Amazon ECR configuration.
  - **image** (string, required) – the full repository name of the application container image to store in Amazon ECR. *Must be in the format <application ID>.<repository name>:<tag>.*
- **releaseInfo** (object) – contains JSON objects with parameters needed to create a pipeline for your target deployment environments.
  - **ECS | EKS** (object) – contains JSON objects representing the environments to target for deployment. The key name specifies the container management service that you are targeting for your application container pipeline. *Key must be "ECS" or "EKS". At least one of the pipeline environments must be enabled.*
    - **beta** (object) –
      - **clusterName** (string, required) – the name of the ECS or EKS cluster to set up in the CloudFormation stack.
      - **serviceName** (string, required*) – the name of the ECS service to set up in the CloudFormation stack.

        ** Applies only to ECS pipelines.*
      - **enabled** (Boolean, required) – a flag indicating whether a beta environment should be configured.
    - **prod** (object) –
      - **clusterName** (string, required) – the name of the ECS or EKS cluster to set up in the CloudFormation stack.
      - **serviceName** (string, required*) – the name of the ECS service to set up in the CloudFormation stack.

        ** Applies only to ECS pipelines.*
      - **enabled** (Boolean, required) – a flag indicating whether a prod environment should be configured.

The following example shows a `pipeline.json` file for a Java application running on Linux. As you can see in the file, the application is running in a beta environment, and there is no prod environment configured yet.

```
{
    "a2CTemplateVersion": "3.1",
    "sourceInfo": {
        "CodeCommit": {
            "repositoryName": "app2container-java-tomcat-6e6f3a87-ecs",
            "branch": "master"
        }
    },
    "releaseInfo": {
        "ECS": {
            "beta": {
                "clusterName": "app2container-java-tomcat-6e6f3a87-ECS-Cluster",
                "serviceName": "app2container-java-tomcat-6e6f3a87-ECS-LBWebAppStack-etc.",
                "enabled": true
            },
            "prod": {
                "clusterName": "",
                "serviceName": "",
                "enabled": false
            }
        }
    }
}
```

Product and service integrations for AWS App2Container

AWS App2Container integrates with an array of AWS services and partner products and services. Use the information in the following sections to help you configure App2Container to integrate with the products and services that you use.

Integrations

Setting up FireLens log file routing for containers with AWS App2Container

When you set up your application containers to use FireLens for Amazon ECS you can route your application logs to CloudWatch, Kinesis Data Streams, or Kinesis Data Firehose for log storage and analytics. After you have configured the FireLens settings in your application analysis and deployment JSON files, App2Container creates the artifacts that you need to deploy your application to Amazon EC2 or AWS Fargate. This includes:

- Creation of initial Kinesis Data Streams or Kinesis Data Firehose streams, if applicable
- Creation of an IAM role with the permissions needed to enable FireLens log routing to the destinations that you have specified
- Deployment artifacts that contain the FireLens parameters that you specified in your JSON configuration files, including the Amazon ECS task definition and AWS CloudFormation template files

For more information about using FireLens for Amazon ECS, see Custom log routing in the *Amazon Elastic Container Service Developer Guide*.

Note
App2Container initially supports FireLens log file routing for Amazon ECS for Linux containers only.

Contents

FireLens log routing for Linux

Before starting these configuration steps, you should have an understanding of the App2Container containerization phases – Initialize, Analyze, Transform, and Deploy. To learn more about the containerization phases and the commands that run during each phase, see the App2Container command reference (p. 45) in this user guide.

Follow these steps to set up log file routing with FireLens for Amazon ECS for your Linux application containers:

FireLens configuration

Prerequisites

Prior to setting up FireLens log routing for your application, you must have completed the following prerequisites:

- You have root access on the application server (and worker machine, if using).
- You successfully completed all of the steps from the Setting up AWS App2Container (p. 5) section of this user guide.
- You have initialized the App2Container environment by successfully running the **init (p. 58)** command.
- The application must be running on the application server, and must have a valid application ID assigned by the **inventory (p. 60)** command.

Step 1: Identify log locations for the container

Run the **analyze (p. 46)** command for your application, and then update the following parameters in your `analysis.json` file:

- Update the `logLocations` array to include a list of log files or directory locations where log files can be picked up for routing with FireLens.
- Set the `enableDynamicLogging` parameter to *true* to map application logs to `stdout` as they are created. If your application appends to specific log files such as `info.log` or `error.log`, set the `enableDynamicLogging` parameter to *false*.

The `analysis.json` file is stored in the application folder, for example: `/root/app2container/java-tomcat-9e8e4799`. For more information on `analysis.json` fields and configuration, see Configuring application containers (p. 17) in the **Configuring your application** section of this user guide.

Example:

The following example shows container parameters in the `analysis.json` file for logging.

```
"containerParameters": {
    ...
    "logFiles": ["error.log", "info.log"],
    "logDirectory": "/var/app/logs/",
    "logLocations": ["error.log", "info.log", "/var/app/logs/"],
    "enableDynamicLogging": true,
    ...
},
```

Step 2: Configure log deployment parameters

Run the **containerize (p. 48)** command, and then edit the `deployment.json` file to set the `fireLensParameters`. The `deployment.json` file is stored in the application folder, for example: `/root/app2container/java-tomcat-9e8e4799`.

There must be at least one valid log destination defined for the `logDestinations` array, with valid values for each of the parameters it contains. For more information on `deployment.json` fields and configuration, including how to target deployment to AWS Fargate with the `deployTarget` parameter, see Configuring container deployment (p. 22) in the **Configuring your application** section of this user guide.

- Set `enableFirelensLogging` to *true*.
- Configure one or more valid `logDestinations` as follows:
  - **service** – the AWS service to route logs to. *Valid values are "cloudwatch", "firehose", and "kinesis".*
  - **regexFilter** (string) – the pattern to match against log content using a Ruby regular expression to determine where to route the log.

    Note
    Ruby regular expressions begin and end with a forward slash, with the pattern to match specified in between the slashes. Patterns often begin with a caret (^), which starts matching at the beginning of the line, and end with a dollar sign ($), which stops matching at the end of the line.
    The `regexFilter` parameter in the `deployment.json` file represents only the matching pattern. Be sure to test your matching pattern using one of the many applications available for your desktop or online, such as Rubular. For more information about Ruby regular expressions, see Mastering Ruby Regular Expressions.
  - **streamName** (string) – the name of the log delivery stream that will be created at the destination.

Examples:

The following example shows FireLens parameters in the `deployment.json` file for logging to a single destination - CloudWatch - using a Ruby regular expression.

```
"fireLensParameters": {
    "enableFireLensLogging": true,
    "logDestinations": [
         {
                "service": "cloudwatch",
                "regexFilter": "^.*INFO.*$",
                "streamName": "Info"
         }
    ]
},
```

This example shows FireLens parameters in the `deployment.json` file for logging to a single destination - Kinesis Data Firehose - using a Ruby regular expression.

```
"fireLensParameters": {
    "enableFireLensLogging": true,
    "logDestinations": [
       {
              "service": "firehose",
              "regexFilter": "^.*INFO.*$",
              "streamName": "Info"
       }
    ]
},
```

This example shows FireLens parameters in the `deployment.json` file for routing separate log files to different destinations in CloudWatch, using Ruby regular expressions.

```
"fireLensParameters": {
    "enableFireLensLogging": true,
    "logDestinations": [
       {
              "service": "cloudwatch",
              "regexFilter": "^.*INFO.*$",
              "streamName": "Info"
       },
       {
              "service": "cloudwatch",
              "regexFilter": "^.*WARNING.*$",
              "streamName": "Warning"
       }
    ]
},
```

Step 3: Validate deployment artifacts

The last step before deployment is to ensure that your Amazon ECS task definitions and AWS CloudFormation templates are configured as expected after running the **generate app-deployment** command, and that your log destinations were created, if applicable.

Note

- Deployment artifacts are stored in the ECS or EKS deployment folder within the application folder that App2Container created for you. For example: `/root/app2container/java-tomcat-9e8e4799`
- If you are routing to CloudWatch, your routing destination is not created prior to deployment.

1. Run the **generate app-deployment (p. 51)** command to generate container deployment artifacts.
2. Verify that the Amazon ECS task definitions include the parameters that you specified and that the values are correct. For an example of FireLens parameters in an Amazon ECS task definition, see Example: Amazon ECS task definition FireLens parameters (p. 31)
3. Verify that the AWS CloudFormation template includes the parameters that you specified and that the values are correct. For an example of FireLens parameters in an AWS CloudFormation template, expand the following section: Example: AWS CloudFormation template FireLens parameters (p. 33)
4. If you are routing logs to Kinesis Data Streams or Kinesis Data Firehose, verify that the streams have been created for you by using the AWS Management Console.

   a. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
   b. From the Amazon Kinesis dashboard, choose **Data streams** or **Delivery streams** from the navigation pane.
   c. Verify that your stream **Status** is `Active`.

Example: Amazon ECS task definition FireLens parameters

This example shows excerpts from an Amazon ECS task definition file that was generated for logging to CloudWatch.

```
"executionRoleArn": arn:aws:iam::
 &lt;YOUR_ACCOUNT_ID&gt;:role/A2CEcsFirelensRole",
"containerDefinitions": [
    {
      ...
      "logConfiguration": {
        "logDriver": "awsfirelens",
        "secretOptions": null,
        "options": {
          "include-pattern": "^.*INFO.*$",
          "log_group_name": "java-tomcat-c770eed9-logs",
          "log_stream_name": "java-tomcat-c770eed9-Info",
          "auto_create_group": "true",
          "region": "us-east-1",
          "Name": "cloudwatch"
        }
      },
      ...
      "name": "java-tomcat-c770eed9"
    },
    {
      "dnsSearchDomains": null,
      "environmentFiles": null,
      "logConfiguration": {
        "logDriver": "awslogs",
        "secretOptions": null,
        "options": {
          "awslogs-group": "/ecs/containerization",
          "awslogs-region": "us-east-1",
          "awslogs-create-group": "true",
          "awslogs-stream-prefix": "firelens"
        }
      },
      ...
      "firelensConfiguration": {
        "type": "fluentbit",
        "options": null
      },
      ...
      "name": "java-tomcat-c770eed9-log-router"
    }
  ],
  ...
  "taskRoleArn": arn:aws:iam::
&lt;YOUR_ACCOUNT_ID&gt;::role/A2CEcsFirelensRole",
  "compatibilities": [
    "EC2",
    "FARGATE"
  ],
  ...
  "requiresAttributes": [
    {
      "targetId": null,
      "targetType": null,
      "value": null,
      "name": "ecs.capability.execution-role-awslogs"
    },
    ...
    {
      "targetId": null,
      "targetType": null,
      "value": null,
      "name": "com.amazonaws.ecs.capability.logging-driver.awsfirelens"
    },
    ...
    {
      "targetId": null,
      "targetType": null,
      "value": null,
      "name": "com.amazonaws.ecs.capability.logging-driver.awslogs"
    },
    ...
    {
      "targetId": null,
      "targetType": null,
      "value": null,
      "name": "ecs.capability.firelens.fluentbit"
    }
  ],
```

Example: AWS CloudFormation template FireLens parameters

This example shows excerpts from an AWS CloudFormation template file that was generated for logging to CloudWatch.

```
Metadata:
  AWS::CloudFormation::Interface:
    ParameterGroups:
...
- Label:
          default: Logging Parameters for the application being deployed, check ecs-lb-
webapp.yml for usage
        Parameters:
          - TaskLogDriver
          - MultipleDests
          - SingleDestName
          - IncludePattern
          - LogGrpName
          - LogStrmName
          - AutoCrtGrp
          - FirehoseStream
          - KinesisStream
          - KinesisAppendNewline
          - FirelensName
          - FirelensImage
          - ConfigType
          - ConfigPath
          - UsingCloudwatchLogs
          - UsingFirehoseLogs
          - UsingKinesisLogs
...
Parameters:
...
# Firelens Parameters for the application being deployed
  TaskLogDriver:
    Type: String
    Default: awsfirelens
  MultipleDests:
    Type: String
    AllowedValues: [true, false]
    Default: false
  SingleDestName:
    Type: String
    Default: cloudwatch
  IncludePattern:
    Type: String
    Default: ^.*INFO.*$
  LogGrpName:
    Type: String
    Default: java-tomcat-c770eed9-logs
  LogStrmName:
    Type: String
    Default: java-tomcat-c770eed9-Info
  AutoCrtGrp:
    Type: String
    Default: true
  FirehoseStream:
    Type: String
    Default: ""
  KinesisStream:
    Type: String
    Default: ""
  KinesisAppendNewline:
    Type: String
    Default: ""
  FirelensName:
    Type: String
    Default: java-tomcat-c770eed9-log-router
  FirelensImage:
    Type: String
    Default: 906394416424.dkr.ecr.us-east-1.amazonaws.com/aws-for-fluent-bit:latest
  ConfigType:
    Type: String
    Default: ""
  ConfigPath:
    Type: String
    Default: ""
  UsingCloudwatchLogs:
    Type: String
    Default: true
  UsingFirehoseLogs:
    Type: String
    Default: false
  UsingKinesisLogs:
    Type: String
    Default: false
...
Rules:
  FirelensSingleCloudwatch:
    RuleCondition: !And
      - !Equals [ !Ref MultipleDests, 'false']
      - !Equals [ !Ref UsingCloudwatchLogs, 'true']
    Assertions:
      - AssertDescription: You cannot use any other firelens destination if a single
 cloudwatch stream is desired
        Assert: !And
          - !Equals [ !Ref UsingFirehoseLogs, 'false']
          - !Equals [ !Ref UsingKinesisLogs, 'false']
          - !Equals [ !Ref SingleDestName, "cloudwatch" ]
          - !Not [ !Equals [ !Ref LogGrpName, "" ]]
          - !Not [ !Equals [ !Ref LogStrmName, "" ]]
          - !Not [ !Equals [ !Ref AutoCrtGrp, "" ]]
  FirelensSingleFirehose:
    RuleCondition: !And
      - !Equals [ !Ref MultipleDests, 'false']
      - !Equals [ !Ref UsingFirehoseLogs, 'true']
    Assertions:
      - AssertDescription: You cannot use any other firelens destination if a single
 firehose stream is desired
        Assert: !And
          - !Equals [ !Ref UsingCloudwatchLogs, 'false']
          - !Equals [ !Ref UsingKinesisLogs, 'false']
          - !Equals [ !Ref SingleDestName, "firehose" ]
          - !Not [ !Equals [ !Ref FirehoseStream, "" ]]
  FirelensSingleKinesis:
    RuleCondition: !And
      - !Equals [ !Ref MultipleDests, 'false']
      - !Equals [ !Ref UsingKinesisLogs, 'true']
    Assertions:
      - AssertDescription: You cannot use any other firelens destination if a single
 kinesis stream is desired
        Assert: !And
          - !Equals [ !Ref UsingCloudwatchLogs, 'false']
          - !Equals [ !Ref UsingFirehoseLogs, 'false']
          - !Equals [ !Ref SingleDestName, "kinesis" ]
          - !Not [ !Equals [ !Ref KinesisStream, "" ]]
          - !Not [ !Equals [ !Ref KinesisAppendNewline, "" ]]
  MultipleDestinations:
    RuleCondition: !Equals [ !Ref MultipleDests, 'true']
    Assertions:
      - AssertDescription: You must supply a configuration file location and filepath if
 multiple firelens destinations are being used
        Assert: !And
          - !Not [ !Equals [ !Ref ConfigType, "" ] ]
          - !Not [ !Equals [ !Ref ConfigPath, "" ] ]
          - !Equals [ !Ref SingleDestName, ""]
          - !Equals [ !Ref IncludePattern, ""]
          - !Equals [ !Ref LogGrpName, ""]
          - !Equals [ !Ref LogStrmName, ""]
          - !Equals [ !Ref AutoCrtGrp, ""]
          - !Equals [ !Ref FirehoseStream, ""]
          - !Equals [ !Ref KinesisStream, ""]
          - !Equals [ !Ref KinesisAppendNewline, ""]
...
Conditions:
...
Resources:
 PrivateAppStack:
    Type: AWS::CloudFormation::Stack
    Condition: DoNotCreatePublicLoadBalancer
    Properties:
      TemplateURL: !Sub 'https://${S3Bucket}.s3.${S3Region}.${AWS::URLSuffix}/
${S3KeyPrefix}/ecs-private-app.yml'
      Tags:
        - Key: "a2c-generated"
          Value: !Sub 'ecs-app-${AWS::StackName}'
      Parameters:
...
        TaskLogDriver: !Ref TaskLogDriver
        MultipleDests: !Ref MultipleDests
        SingleDestName: !Ref SingleDestName
        IncludePattern: !Ref IncludePattern
        LogGrpName: !Ref LogGrpName
        LogStrmName: !Ref LogStrmName
        AutoCrtGrp: !Ref AutoCrtGrp
        FirehoseStream: !Ref FirehoseStream
        KinesisStream: !Ref KinesisStream
        KinesisAppendNewline: !Ref KinesisAppendNewline
        FirelensName: !Ref FirelensName
        FirelensImage: !Ref FirelensImage
        ConfigType: !Ref ConfigType
        ConfigPath: !Ref ConfigPath
        UsingCloudwatchLogs: !Ref UsingCloudwatchLogs
        UsingFirehoseLogs: !Ref UsingFirehoseLogs
        UsingKinesisLogs: !Ref UsingKinesisLogs
...
```

Step 4: Deploy your application to Amazon ECS

Deploy your application using the generate app-deployment (p. 51) command with the `--deploy` option.

```
$ sudo app2container generate app-deployment --deploy --application id java-tomcat-9e8e4799
# AWS prerequisite check succeeded
# Docker prerequisite check succeeded
# Created ECR Repository
# Registered ECS Task Definition with ECS
# Uploaded CloudFormation resources to S3 Bucket: app2container-example
# Generated CloudFormation Master template at: /root/app2container/java-tomcat-9e8e4799/
EcsDeployment/ecs-master.yml
# Initiated CloudFormation stack creation. This may take a few minutes. Please visit the
 AWS CloudFormation Console to track progress.
ECS deployment successful for application java-tomcat-9e8e4799

The URL to your Load Balancer Endpoint is:
<your endpoint>.us-east-1.elb.amazonaws.com
Successfully created ECS stack app2container-java-tomcat-9e8e4799-ECS. Check the AWS
 CloudFormation Console for additional details.
```

Alternatively, you can deploy your application's AWS CloudFormation template using the AWS CLI as follows.

```
$ sudo aws cloudformation deploy --template-file /root/app2container/java-tomcat-9e8e4799/
EcsDeployment/ecs-master.yml --capabilities CAPABILITY_NAMED_IAM --stack-name
 app2container-java-tomcat-9e8e4799-ECS
```

Step 5: Verify log routing

After you deploy your application to Amazon ECS, you can verify that your logs are routing to their intended destinations.

Security in AWS App2Container

Security at AWS is the highest priority. As an AWS customer using AWS App2Container and tools such as Amazon ECR, Amazon ECS, and Amazon EKS, you benefit from data centers and network architectures that are 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 AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Amazon EC2, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

Contents

Data protection in App2Container

Installation packages for App2Container are built and signed by an internal build pipeline and uploaded to a private Amazon S3 bucket. The packages are available for download using links that are shared in the download instructions.

App2Container is installed directly onto application servers and worker machines and does much of its work locally, using direct commands entered by a server administrator in the command line interface. It requires administrator credentials to operate. As a result, the artifacts that it creates can only be accessed by an administrator. This protects against malicious users trying to access these application artifacts on the server without having the appropriate credentials.

Much of the configuration for containerization with App2Container is stored in JSON files in the application directory. Before configuration entries from JSON files are used, App2Container performs input validation to prevent malicious code injection and to ensure that command execution is not altered in any way.

App2Container ensures that no sensitive fields are logged. The logs themselves reside locally on the application server or worker machine.

If you consented to metric collection when you ran the **init** command, App2Container collects metrics and stores them in a private location for the service. Metrics do not contain sensitive or identifying data, and are kept in accordance with GDPR compliance guidelines for data retention. For a list of the metrics that App2Container collects, see the **init** (p. 58) command.

Data encryption

App2Container communicates with AWS services using standard APIs when retrieving artifacts from Amazon S3 or pushing Docker containers to service endpoints in the AWS container management suite

(Amazon ECR, Amazon ECS, and Amazon EKS). It works with AWS CloudFormation and AWS CodeStar services to generate and deploy relevant container and lifecycle artifacts using their standard APIs.

Encryption at rest

- App2Container installation packages are kept in a private Amazon S3 bucket with encryption enabled.
- Application artifacts can optionally be uploaded into Amazon S3 buckets. Enable encryption for your Amazon S3 bucket to enforce data encryption.

Encryption in transit

- App2Container installation packages are kept in a private Amazon S3 bucket, which requires secure download using the HTTPS protocol using links provided for each package.
- App2Container uses standard AWS APIs for the services it interacts with, including Amazon ECR, Amazon ECS, Amazon EKS, AWS CloudFormation, CodePipeline, and Amazon S3. AWS APIs use HTTPS as their default communication protocol.

Internetwork traffic privacy

App2Container does not store passwords, keys, or other secrets or customer-sensitive material. App2Container also ensures that no sensitive fields are contained in application logs.

Identity and access management in App2Container

Your AWS security credentials identify you to AWS and grant you access to your AWS resources. For example, they can allow you to access artifacts saved to an Amazon S3 bucket. You can use features of AWS Identity and Access Management (IAM) to allow other users, services, and applications to use specific resources in your AWS account without sharing your security credentials. You can choose to allow full use or limited use of your AWS resources.

If you are the owner of the AWS account and use AWS as the root user, we strongly recommend that you create an IAM admin user to use for access to your AWS resources. See Creating Your First IAM Admin User and Group in the *IAM User Guide* to set up your own access before setting up any other IAM users who need to use App2Container.

By default, IAM users don't have permission to create or modify resources. To allow IAM users to create or modify resources and perform tasks, you must create IAM policies that grant permission to use the specific resources and API actions that they need. For more information about IAM policies, see Policies and Permissions in the *IAM User Guide*.

IAM groups and roles are a flexible way to manage permissions across multiple users. When you assign a user to a group or when your user assumes a role, that user inherits the group's or role's permissions, and is allowed or denied permission to perform the specified tasks on the specified resources. You can assign multiple users to the same group, and a role can be assumed by authorized users. While groups and roles both serve the purpose of granting access to resources, roles are more task-oriented, and assuming a role provides you with temporary security credentials for your role session.

IAM security best practices
Follow these top four security best practices when setting up your IAM resources. For more information and additional best practices, see Security Best Practices in IAM in the *IAM User Guide*.

1. **Lock away your AWS account root user access keys**

Protect your root user access key like you would your credit card numbers or any other sensitive secret, and only use your root user account for necessary account and service management tasks.

2. **Create individual IAM users**

   Don't use your AWS account root user credentials to access AWS, and don't give your credentials to anyone else. Instead, create individual users for anyone who needs access to your AWS account.

3. **Use groups or roles to assign permissions to IAM Users**

   Instead of defining permissions for individual IAM users, it's usually more convenient to create groups that relate to job functions (administrators, developers, accounting, etc.) or roles that relate to specific tasks.

4. **Grant least privilege**

   When you create IAM policies, follow the standard security advice of granting *least privilege*, or granting only the permissions required to perform a task. Determine what users (and roles) need to do and then craft policies that allow them to perform only those tasks.

We recommend that you create a general purpose IAM group that can run all of the commands *except* commands that are run with the `--deploy` option.

If you plan to use App2Container to deploy your containers or create pipelines, then you should create a separate IAM user for deployments. The deployment user needs to be able to create or update AWS objects for container management services (Amazon ECR with Amazon ECS or Amazon EKS), and to create pipelines with AWS CodeStar services. This requires elevated permissions that should only be used for deployment.

Set up IAM resources for App2Container

- Create IAM resources for general use (p. 39)
- Create IAM resources for deployment (p. 44)

Create IAM resources for general use

Follow best practices by using the following steps to create an IAM group with access to perform specific tasks, using specific resources, and to assign users to the group. Alternatively, you can choose to assign an inline policy to the IAM user who will run **app2container** commands.

1. **Create a customer managed IAM policy**

   You can create a customer managed IAM policy for your general purpose user or group, using one of the example policies (p. 40) on this page after you have customized the JSON to refer to your resources. To create a policy using the AWS console, see Creating policies on the JSON tab in the *IAM User Guide*. To create a policy using the AWS CLI, use the **create-policy** command.

2. **Create IAM users and a group**

   Every user who will run **app2container** commands needs to have an IAM user created for accessing AWS resources under your account. To follow best practices, you can create an IAM group with the policy attached, and assign users to it.

   To create an IAM user, see Creating an IAM User in Your AWS Account in the *IAM User Guide*. Be sure to select programmatic access to AWS when you create the IAM user.

   Perform the following steps to create an IAM group and assign users to it.

a. To create an IAM group, see Creating IAM Groups in the *IAM User Guide*.
b. Ensure that every person who will run **app2container** commands has an IAM user defined for AWS access.
c. To assign the users to the group you created in step 1a, see Adding Permissions to a User (Console), or Adding and Removing a User's Permissions (AWS CLI or AWS API) in the *IAM User Guide*.

3. **Save your AWS access keys**

   Save the access keys for your new or existing IAM user in a safe place. You'll need them to configure your AWS profile (p. 6) as part of getting set up for App2Container.

4. **Attach or assign the policy**

   Use one of the following methods to assign permissions to your IAM users.

   - **Attach the policy to the IAM group**

     Attach the policy you created in step 1 to the group you created in step 2. See Attaching a Policy to an IAM Group in the *IAM User Guide*.

   - **Embed the policy inline for an IAM user**

     Embed the policy you created in step 1 inline for your IAM user. See the section that begins with "To embed an inline policy" in Adding Permissions to a User (Console), or Adding and Removing a User's Permissions (AWS CLI or AWS API) in the *IAM User Guide*.

Example IAM policies

You can use one of the following templates as a starting point to configure the access that an IAM user needs to use App2Container to containerize your applications. If you plan to store extracts or other resources in Amazon S3, your policy must grant access to the buckets.

IAM policy for Amazon ECS

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SectionForS3Access",
            "Action": [
                "s3:DeleteObject",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:HeadBucket",
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Effect": "Allow",
            "Resource": "<user-provided-bucket-ARN>"
        },
        {
            "Sid": "SectionForS3ReadAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketAcl"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Sid": "SectionForECRAccess",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchDeleteImage",
                "ecr:BatchGetImage",
                "ecr:CompleteLayerUpload",
                "ecr:CreateRepository",
                "ecr:DeleteRepository",
                "ecr:DescribeImages",
                "ecr:DescribeRepositories",
                "ecr:GetAuthorizationToken",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetRepositoryPolicy",
                "ecr:InitiateLayerUpload",
                "ecr:ListImages",
                "ecr:PutImage",
                "ecr:TagResource",
                "ecr:UntagResource",
                "ecr:UploadLayerPart"
            ],
            "Effect": "Allow",
            "Resource": "<resource-ARNs>"
        },
        {
            "Sid": "SectionForECSWriteAccess",
            "Action": [
                "ecs:CreateCluster",
                "ecs:CreateService",
                "ecs:CreateTaskSet",
                "ecs:DeleteCluster",
                "ecs:DeleteService",
                "ecs:DeleteTaskSet",
                "ecs:DeregisterTaskDefinition",
                "ecs:Poll",
                "ecs:RegisterContainerInstance",
                "ecs:RegisterTaskDefinition",
                "ecs:RunTask",
                "ecs:StartTask",
                "ecs:StopTask",
                "ecs:SubmitContainerStateChange",
                "ecs:SubmitTaskStateChange",
                "ecs:UpdateContainerInstancesState",
                "ecs:UpdateService",
                "ecs:UpdateServicePrimaryTaskSet",
                "ecs:UpdateTaskSet"
            ],
            "Effect": "Allow",
            "Resource": "<resource-ARNs>"
        },
        {
            "Sid": "SectionForPassRoleToECS",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "<ARN for ecsTaskExecutionRole>"
        },
        {
            "Sid": "SectionForECSReadAccess",
            "Action": [
                "ecs:DescribeClusters",
                "ecs:DescribeContainerInstances",
                "ecs:DescribeServices",
                "ecs:DescribeTaskDefinition",
                "ecs:DescribeTaskSets",
                "ecs:DescribeTasks",
                "ecs:ListClusters",
                "ecs:ListContainerInstances",
                "ecs:ListServices",
                "ecs:ListTaskDefinitionFamilies",
                "ecs:ListTaskDefinitions",
                "ecs:ListTasks"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "SectionForCodeCommitAccess",
            "Effect": "Allow",
            "Action": [
                "codecommit:GetRepository",
                "codecommit:GetBranch",
                "codecommit:CreateRepository",
                "codecommit:CreateCommit",
                "codecommit:TagResource"
            ],
            "Resource": "arn:aws:codecommit:*:*:*"
        },
        {
            "Sid": "SectionForByoVPC",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInternetGateways",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs"
            ],
            "Resource": "*"
        },
        {
          "Sid": "SectionForEC2",
          "Effect": "Allow",
          "Action": [
             "ec2:DescribeKeyPairs",
             "ec2:CreateKeyPair",
             "ec2:DescribeAvailabilityZones"
          ],
          "Resource": "*"
        },
        {
            "Sid": "SectionForMetricsService",
            "Effect": "Allow",
            "Action": "execute-api:invoke",
            "Resource": "arn:aws:execute-api:us-east-1:*:*/prod/POST/put-metric-data"
        }
    ]
}
```

IAM policy for Amazon EKS

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SectionForS3Access",
            "Action": [
                "s3:DeleteObject",
                "s3:GetBucketAcl",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:HeadBucket",
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Effect": "Allow",
            "Resource": "<user-provided-bucket-ARN>"
        },
        {
            "Sid": "SectionForS3ReadAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketAcl"
            ],
            "Resource": "arn:aws:s3:::*"
        },
        {
            "Sid": "SectionForECRAccess",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchDeleteImage",
                "ecr:BatchGetImage",
                "ecr:CompleteLayerUpload",
                "ecr:CreateRepository",
                "ecr:DeleteRepository",
                "ecr:DescribeImages",
                "ecr:DescribeRepositories",
                "ecr:GetAuthorizationToken",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetRepositoryPolicy",
                "ecr:InitiateLayerUpload",
                "ecr:ListImages",
                "ecr:PutImage",
                "ecr:TagResource",
                "ecr:UntagResource",
                "ecr:UploadLayerPart"
            ],
            "Effect": "Allow",
            "Resource": "<resource-ARNs>"
        },
        {
            "Sid": "SectionForCodeCommitAccess",
            "Effect": "Allow",
            "Action": [
                "codecommit:GetRepository",
                "codecommit:GetBranch",
                "codecommit:CreateRepository",
                "codecommit:CreateCommit",
                "codecommit:TagResource"
            ],
            "Resource": "arn:aws:codecommit:*:*:*"
        },
        {
            "Sid": "SectionForByoVPC",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInternetGateways",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs"
            ],
            "Resource": "*"
        },
        {
          "Sid": "SectionForEC2",
          "Effect": "Allow",
          "Action": [
             "ec2:DescribeKeyPairs",
             "ec2:CreateKeyPair",
             "ec2:DescribeAvailabilityZones"
          ],
          "Resource": "*"
        },
        {
            "Sid": "SectionForMetricsService",
            "Effect": "Allow",
            "Action": "execute-api:invoke",
            "Resource": "arn:aws:execute-api:us-east-1:*:*/prod/POST/put-metric-data"
        }
    ]
}
```

Create IAM resources for deployment

The **AdministratorAccess** policy grants an IAM user full access to AWS. Therefore, IAM users with this policy can deploy a containerized application using any of the AWS services for deployment that are supported by App2Container.

1. **Create an IAM user**

   You can create an IAM user with full access to AWS API actions and resources. Be sure to grant the user programmatic access to AWS and to attach the **AdministratorAccess** policy. For more information, see Creating IAM users in the *IAM User Guide*.
2. **Save your AWS access keys**

   Save the access keys for the IAM user in a safe place. You'll need them to configure your AWS profile (p. 6) as part of getting set up for App2Container.

Update management for App2Container

App2Container detects what version of the CLI you are using when you run a command. It notifies you if there are published updates available. You can install the latest version of App2Container using the upgrade (p. 61) command.

App2Container command reference

AWS App2Container is a command line tool that transforms supported legacy applications running on physical servers or virtual machines into applications that run in Docker containers on Amazon ECS or Amazon EKS.

> **Note**
> Running App2Container commands on a Linux server requires elevated permissions. Prefix the command syntax with **sudo**, or run the **sudo su** command one time when you log in before running the commands as shown in the syntax for the commands linked below.

Containerization phases

The containerization process has several phases.

Phases

Initialize

The init command performs one-time initialization tasks for App2Container. Run this command before you run any other App2Container commands.

init (p. 58)

Run the **init** command to configure the AWS App2Container workspace on your application servers and worker machines.

Analyze

After setup and initialization have been completed on your application servers, you can begin the analyze phase. Run these commands on your existing application servers even if you plan to use a worker machine for the containerization steps.

inventory (p. 60)

Run the **inventory** command to produce an inventory of applications that are running on your application servers and assign each one a unique ID to use when you run other commands.

analyze (p. 46)

Run the **analyze** command to analyze your running applications and identify dependencies that are required for containerization. This command creates the `analysis.json` file that feeds into the Transform phase commands.

Transform

The transform phase creates containers for your applications that have gone through analysis.

extract (p. 50)

Run the **extract** command on your application server or worker machine to generate an application archive based on the `analysis.json` file created by the **analyze** command. Transfer the archive to the worker machine for the remaining steps that require the operating system to support containers.

containerize (p. 48)

Run the **containerize** command to do the following:

- Extract application artifacts or read from an extract archive
- Generate Docker container artifacts for your application using the `analysis.json` file that was generated by the **analyze** command
- Create the `deployment.json` file for input to the **generate app-deployment** command

Deploy

The deploy phase consists of running the **generate app-deployment** command and configuring the artifacts that are used to deploy to your container management environments (Amazon ECR with Amazon ECS or Amazon EKS). You can optionally deploy to target environments, and create a CI/CD pipeline using the **generate pipeline** command.

Each command in this phase has two steps — one to generate artifacts and one to deploy.

generate app-deployment (p. 51)

Step 1: Generate deployment artifacts

Run the **generate app-deployment** command to generate container deployment artifacts for target container management environments using the `deployment.json` file generated by the **containerize** command, and to create the `pipeline.json` file for input to the **generate pipeline** command.

The deployment step is optional, depending on how and where you choose to deploy.

Step 2: Deploy to target environment

After you have reviewed the artifacts from Step 1 and have applied any necessary changes, run the **generate app-deployment** command with the `--deploy` option to deploy the container to your target environment.

generate pipeline (p. 54) (optional)

Step 1: Generate CI/CD pipeline artifacts

Run the **generate pipeline** command to generate CI/CD pipeline artifacts for automated container deployment using AWS CodeStar services and the `deployment.json` file generated by the **generate app-deployment** command.

Step 2: Deploy the CI/CD pipeline

After you have reviewed the artifacts from Step 1 and have applied any necessary changes, run the **generate pipeline** command with the `--deploy` option to deploy your container pipeline.

app2container analyze command

Analyzes the specified application and generates a report.

Syntax

```
app2container analyze --application-id id [--help]
```

Options

--application-id *id*

The application ID. After you run the inventory (p. 60) command, you can find the application ID in the `inventory.json` file in one of the following locations:

- **Linux:** `/root/inventory.json`
- **Windows:** `C:\Users\Administrator\AppData\Local\.app2container-config\inventory.json`

--help

Displays the command help.

Output

The **analyze** command creates files and directories for each application. Output varies slightly, depending on your application language and the application server operating system.

The application directory is created in the output location that you specified when you ran the **init** command. Each application has its own directory named for the application ID. The directory contains analysis output and editable application configuration files. The files are stored in subdirectories that match the application structure on the server.

An `analysis.json` file is created for the application that is specified in the `--application-id` parameter. The file contains information about the application found during analysis, and configurable fields for container settings. See analysis.json file (p. 17) for more information about configurable fields, and for an example of what the file looks like.

For .NET applications, App2Container detects connection strings and produces the `report.txt` file. The report location is specified in the `analysis.json` file, in the `reportPath` attribute of the `analysisInfo` section. You can use this report to identify the changes you need to make in application configuration files to connect your application container to new database endpoints, if needed. The report also contains the locations of other configuration files that might need changes.

Examples

Choose the operating system tab for the application server or worker machine where you run the command.

Linux

The following example shows the **analyze** command with application id and no additional options.

```
$ sudo app2container analyze --application id java-tomcat-9e8e4799
# Created artifacts folder /root/app2container/java-tomcat-9e8e4799
# Generated analysis data in /root/app2container/java-tomcat-9e8e4799/analysis.json
Analysis successful for application java-tomcat-9e8e4799
Please examine the application analysis file at /root/app2container/java-
tomcat-9e8e4799/analysis.json,
make appropriate edits and initiate containerization using "app2container containerize
 --application-id java-tomcat-9e8e4799
```

Windows

The following example shows the **analyze** command with application id and no additional options.

```
PS> app2container analyze --application id iis-smarts-51d2dbf8
# Created artifacts folder C:\Users\Administrator\AppData\Local\app2container\iis-
smarts-51d2dbf8
# Generated analysis data in C:\Users\Administrator\AppData\Local\app2container\iis-
smarts-51d2dbf8\analysis.json
Analysis successful for application iis-smarts-51d2dbf8
Please examine the application analysis file at C:\Users\Administrator\AppData\Local
\app2container\iis-smarts-51d2dbf8\analysis.json,
make appropriate edits and initiate containerization using "app2container containerize
 --application-id iis-smarts-51d2dbf8
```

app2container containerize command

Generates a container image for the specified application or application archive.

Syntax

```
app2container containerize {--application-id id | --input-archive extraction-file} [--help]
```

Options

--application-id *id*

The application ID. After you run the inventory (p. 60) command, you can find the application ID in the `inventory.json` file in one of the following locations:

- **Linux:** `/root/inventory.json`
- **Windows:** `C:\Users\Administrator\AppData\Local\.app2container-config\inventory.json`

--build-only

Builds container images based on the existing `Dockerfile` and artifacts.

--force

Bypasses the disk space prerequisite check.

--input-archive *extraction-file*

The file path or Amazon S3 key (for example, s3://*bucket*/*archive-key*) for the application archive. If you specify an application archive, the command downloads and opens the archive, and then builds the container image.

--help

Displays the command help.

Output

This command generates a `Dockerfile`, a container image, and a `deployment.json` file that you can use with the generate app-deployment (p. 51) command.

It also generates a `Dockerfile.update` file that you can use to make updates to your containerized application. The generate pipeline (p. 54) command adds this Dockerfile to your CodeCommit repository and deploys updates to your CodePipeline infrastructure.

See deployment.json file (p. 22) for more information about configuration, and for an example of what the `deployment.json` file looks like.

Examples

Choose the operating system tab for the application server or worker machine where you run the command.

Linux

The following example shows the **containerize** command with application id and no additional options.

```
$ sudo app2container containerize --application id java-tomcat-9e8e4799
# AWS pre-requisite check succeeded
# Docker pre-requisite check succeeded
# Extracted container artifacts for application
# Entry file generated
# Dockerfile generated under /root/app2container/java-tomcat-9e8e4799/Artifacts
# Generated dockerfile.update under /root/app2container/java-tomcat-9e8e4799/Artifacts
# Generated deployment file at /root/app2container/java-tomcat-9e8e4799/deployment.json
Containerization successful. Generated docker image java-tomcat-9e8e4799

You're all set to test and deploy your container image.

Next Steps:
1. View the container image with \"docker images\" and test the application.
2. When you're ready to deploy to AWS, please edit the deployment file as needed at /
root/app2container/java-tomcat-9e8e4799/deployment.json.
3. Generate deployment artifacts using app2container generate app-deployment --
application-id java-tomcat-9e8e4799
Please use "docker images" to view the generated container image.
```

The following example shows the **containerize** command with the `--input-archive` option.

```
$ sudo app2container containerize --input-archive /var/aws/java-tomcat-9e8e4799/java-
tomcat-9e8e4799-extraction.tar
```

Windows

The following example shows the **containerize** command with application id and no additional options.

```
PS> app2container containerize --application id iis-smarts-51d2dbf8
# AWS pre-requisite check succeeded
# Docker pre-requisite check succeeded
# Extracted container artifacts for application
# Entry file generated
# Dockerfile generated under C:\Users\Administrator\AppData\Local\app2container\iis-
smarts-51d2dbf8\Artifacts
# Generated dockerfile.update under C:\Users\Administrator\AppData\Local\app2container
\iis-smarts-51d2dbf8\Artifacts
# Generated deployment file at C:\Users\Administrator\AppData\Local\app2container\iis-
smarts-51d2dbf8\deployment.json
Containerization successful. Generated docker image iis-smarts-51d2dbf8

You're all set to test and deploy your container image.
```

```
Next Steps:
1. View the container image with \"docker images\" and test the application.
2. When you're ready to deploy to AWS, please edit the deployment file as needed at C:
\Users\Administrator\AppData\Local\app2container\iis-smarts-51d2dbf8\deployment.json.
3. Generate deployment artifacts using app2container generate app-deployment --
application-id iis-smarts-51d2dbf8
Please use "docker images" to view the generated container image.
```

The following example shows the **containerize** command with the `--input-archive` option.

```
PS> app2container containerize --input-archive archive C:\Users\Administrator\Downloads
\iis-smarts-51d2dbf8.zip
# AWS pre-requisite check succeeded
# Docker pre-requisite check succeeded
# Dockerfile generated under C:\Users\Administrator\AppData\Local\app2container\iis-
smarts-51d2dbf8\Artifacts
# Generated dockerfile.update under C:\Users\Administrator\AppData\Local\app2container
\iis-smarts-51d2dbf8\Artifacts
# Generated deployment file at C:\Users\Administrator\AppData\Local\app2container\iis-
smarts-51d2dbf8\deployment.json
Containerization successful. Generated docker image iis-smarts-51d2dbf8

You're all set to test and deploy your container image.

Next Steps:
1. View the container image with \"docker images\" and test the application.
2. When you're ready to deploy to AWS, please edit the deployment file as needed at C:
\Users\Administrator\AppData\Local\app2container\iis-smarts-51d2dbf8\deployment.json.
3. Generate deployment artifacts using app2container generate app-deployment --
application-id iis-smarts-51d2dbf8
To have gMSA related artifacts generated with CloudFormation, please edit gMSAParams
 inside deployment file.
Otherwise look at C:\Users\Administrator\AppData\Local\app2container\iis-
smarts-51d2dbf8\Artifacts\WindowsAuthSetupInstructions.md for setup instructions on
 Windows Authentication
Please use "docker images" to view the generated container image.
```

app2container extract command

Generates an application archive for the specified application. Before you call this command, you must call the analyze (p. 46) command.

Syntax

```
app2container extract --application-id id [--output s3] [--help]
```

Options

--application-id *id*

The application ID. After you run the inventory (p. 60) command, you can find the application ID in the `inventory.json` file in one of the following locations:

- **Linux:** `/root/inventory.json`
- **Windows:** `C:\Users\Administrator\AppData\Local\.app2container-config\inventory.json`

--force

Bypasses the disk space prerequisite check.

--output s3

If specified, this option writes the archive file to an Amazon S3 bucket that you specified when you ran the **init** command.

--help

Displays the command help.

Output

This command creates an archive file. When you use the `--output s3` option, the archive is written to the Amazon S3 bucket that you specified when you ran the **init** command. Otherwise, the archive is written to the output location that you specified when you ran the **init** command.

Examples

Choose the operating system tab for the application server or worker machine where you run the command.

Linux

The following example shows the **extract** command with application id and no additional options.

```
$ sudo app2container extract --application id java-tomcat-9e8e4799
# Extracted container artifacts for application
# Application archive file created at: /root/app2container/java-tomcat-9e8e4799/java-
tomcat-9e8e4799-extraction.tar
Extraction successful for application java-tomcat-9e8e4799

Please transfer this tar file to your worker machine and run, "app2container
 containerize --input-archive <extraction-tar-filepath>"
```

Windows

The following example shows the **extract** command with application id and no additional options.

```
PS> app2container extract --application id iis-smarts-51d2dbf8
# Extracted container artifacts for application
Extraction successful for application iis-smarts-51d2dbf8
```

app2container generate app-deployment command

When you initially run this command, it generates the artifacts needed to deploy your application container in AWS. After you review and configure the artifacts, you can run this command again with the `--deploy` option to deploy to your target container management environment (Amazon ECR with Amazon ECS or Amazon EKS).

This command accesses AWS resources when you use it to generate deployment artifacts, and again when you deploy. The IAM user with administrator access that you created during security setup is required to run the command with the `--deploy` option. See Identity and access management in App2Container (p. 38) for more information about setting up IAM users for App2Container.

The command uses the `deployment.json` file that is generated by the containerize (p. 48) command. You can edit the `deployment.json` file to specify the following:

- An image repository name for Amazon ECR
- Task definition parameters for Amazon ECS
- The Kubernetes app name

Syntax

```
app2container generate app-deployment --application-id id [--deploy] [--help]
```

Options

--application-id *id*

The application ID. After you run the inventory (p. 60) command, you can find the application ID in the `inventory.json` file in one of the following locations:

- **Linux:** `/root/inventory.json`
- **Windows:** `C:\Users\Administrator\AppData\Local\.app2container-config\inventory.json`

--deploy

After you have reviewed and made updates to the deployment artifacts, use this option to deploy to your target container management environment (Amazon ECR with Amazon ECS or Amazon EKS).

--help

Displays the command help.

Output

The **generate app-deployment** command generates container deployment artifacts for you to review and edit. When the artifacts are ready to deploy, you can run this command with the `--deploy` option to deploy to your target container management environment.

The lists below show what artifacts are generated and what tasks are performed when you run the **generate app-deployment** command.

1. Generate container deployment artifacts: **generate app-deployment**
   - Checks for AWS and Docker prerequisites
   - Creates an Amazon ECR repository
   - Tasks for Amazon ECS
     - Generates an Amazon ECS task definition template
   - Tasks for Amazon EKS
     - Generates a Kubernetes `deployment.yaml` file that you can use with Amazon EKS or the **kubectl** command
   - Generates an AWS CloudFormation Master template
   - Generates a `pipeline.json` file
2. Deploy container deployment artifacts: **generate app-deployment** `--deploy`

- Checks for AWS and Docker prerequisites
- Registers the container with the Amazon ECR repository that was created when you ran the command without the `--deploy` option.
- Tasks for Amazon ECS
  - Registers the task definition with Amazon ECS
- Uploads AWS CloudFormation resources to an Amazon S3 bucket
- Creates a AWS CloudFormation stack

See pipeline.json file (p. 26) for more information about pipeline configuration, and for an example `deployment.json` file.

Examples

Choose the operating system tab for the application server or worker machine where you run the command.

Linux

The following example shows the **generate app-deployment** command with application id.

```
$ sudo app2container generate app-deployment --application id java-tomcat-9e8e4799
# AWS pre-requisite check succeeded
# Docker pre-requisite check succeeded
# Created ECR Repository
# Registered ECS Task Definition with ECS
# Uploaded CloudFormation resources to S3 Bucket: app2container/-example
# Generated CloudFormation Master template at: /root/app2container/java-
tomcat-9e8e4799/EcsDeployment/ecs-master.yml
ECS CloudFormation templates and additional deployment artifacts generated successfully
 for application java-tomcat-9e8e4799

You're all set to use AWS CloudFormation to manage your application stack.

Next Steps:
1. Edit the cloudformation template as necessary.
2. Create an application stack using the AWS CLI or the AWS Console. AWS CLI command:
aws cloudformation deploy --template-file /root/app2container/java-tomcat-9e8e4799/
EcsDeployment/ecs-master.yml --capabilities CAPABILITY_NAMED_IAM --stack-name
 app2container-java-tomcat-9e8e4799-ECS
3. Setup a pipeline for your application stack using app2container:
app2container generate pipeline --application-id java-tomcat-9e8e4799
```

The following example shows the **generate app-deployment** command with application id and the `--deploy` option.

```
$ sudo app2container generate app-deployment --deploy --application id java-
tomcat-9e8e4799
# AWS prerequisite check succeeded
# Docker prerequisite check succeeded
# Created ECR Repository
# Registered ECS Task Definition with ECS
# Uploaded CloudFormation resources to S3 Bucket: app2container-example
# Generated CloudFormation Master template at: /root/app2container/java-
tomcat-9e8e4799/EcsDeployment/ecs-master.yml
# Initiated CloudFormation stack creation. This may take a few minutes. Please visit
 the AWS CloudFormation Console to track progress.
ECS deployment successful for application java-tomcat-9e8e4799

The URL to your Load Balancer Endpoint is:
```

```
<your endpoint>.us-east-1.elb.amazonaws.com
Successfully created ECS stack app2container-java-tomcat-9e8e4799-ECS. Check the AWS
 CloudFormation Console for additional details.
```

Windows

The following example shows the **generate app-deployment** command with application id.

```
PS> app2container generate app-deployment --application id iis-smarts-51d2dbf8
# AWS pre-requisite check succeeded
# Docker pre-requisite check succeeded
# Created ECR Repository
# Registered ECS Task Definition with ECS
# Uploaded CloudFormation resources to S3 Bucket: app2container\-testing
# Generated CloudFormation Master template at: C:\Users\Administrator\AppData\Local
\app2container\iis-smarts-51d2dbf8\EcsDeployment\ecs-master.yml
ECS CloudFormation templates and additional deployment artifacts generated successfully
 for application iis-smarts-51d2dbf8

You're all set to use AWS CloudFormation to manage your application stack.

Next Steps:
1. Edit the cloudformation template as necessary.
2. Create an application stack using the AWS CLI or the AWS Console. AWS CLI command:
aws cloudformation deploy --template-file C:\Users\Administrator\AppData\Local
\app2container\iis-smarts-51d2dbf8\EcsDeployment\ecs-master.yml --capabilities
 CAPABILITY_NAMED_IAM --stack-name app2container-iis-smarts-51d2dbf8-ECS
3. Setup a pipeline for your application stack using app2container:
app2container generate pipeline --application-id iis-smarts-51d2dbf8
```

The following example shows the **generate app-deployment** command with application id and the `--deploy` option.

```
PS> app2container generate app-deployment --deploy --application id iis-smarts-51d2dbf8
# AWS prerequisite check succeeded
# Docker prerequisite check succeeded
# Created ECR Repository
# Registered ECS Task Definition with ECS
# Uploaded CloudFormation resources to S3 Bucket: app2container-example
# Generated CloudFormation Master template at: C:\Users\Administrator\AppData\Local
\app2container\iis-smarts-51d2dbf8\EcsDeployment\ecs-master.yml
# Initiated CloudFormation stack creation. This may take a few minutes. Please visit
 the AWS CloudFormation Console to track progress.
ECS deployment successful for application iis-smarts-51d2dbf8

The URL to your Load Balancer Endpoint is:
<your endpoint>.us-east-1.elb.amazonaws.com
Successfully created ECS stack app2container-iis-smarts-51d2dbf8-ECS. Check the AWS
 CloudFormation Console for additional details.
```

app2container generate pipeline command

When you first run this command, it generates the artifacts needed to create a CI/CD pipeline with AWS CodeStar, based on your application deployment settings and artifacts. After you review and configure the artifacts, you can run this command again with the `--deploy` option to create your pipeline.

This command accesses AWS resources when you use it to generate CI/CD pipeline artifacts, and again when you deploy. The IAM user with administrator access that you created during security setup is required to run the command with the `--deploy` option. See Identity and access management in App2Container (p. 38) for more information about setting up IAM users for App2Container.

The command uses the `pipeline.json` file that is generated by the generate app-deployment (p. 51) command. You can edit the `pipeline.json` file to specify your container repository and target environments for Amazon ECS or Amazon EKS.

Syntax

```
app2container generate pipeline --application-id id [--deploy] [--help]
```

Options

--application-id *id*

The application ID. After you run the inventory (p. 60) command, you can find the application ID in the `inventory.json` file in one of the following locations:

- **Linux:** `/root/inventory.json`
- **Windows:** `C:\Users\Administrator\AppData\Local\.app2container-config\inventory.json`

--deploy

After you have reviewed and made updates to the pipeline artifacts, use this option to create your CI/CD pipeline.

--help

Displays the command help.

Output

The **generate pipeline** command generates pipeline artifacts for you to review and edit. When the artifacts are ready to deploy, you can run this command with the `--deploy` option to create your pipeline.

The lists below show what artifacts are generated and what tasks are performed when you run the **generate pipeline** command.

1. Generate CI/CD artifacts: **generate pipeline**
   - Checks for AWS and Docker prerequisites
   - Creates a CodeCommit repository
   - Generates a buildspec file
   - Generates AWS CloudFormation templates
2. Deploy CI/CD artifacts: **generate pipeline** `--deploy`
   - Checks for AWS and Docker prerequisites
   - Commits files to a CodeCommit repository
   - Creates an AWS CloudFormation stack

Examples

Choose the operating system tab for the application server or worker machine where you run the command.

Linux

The following example shows the **generate pipeline** command with application id.

```
$ sudo app2container generate pipeline --application id java-tomcat-9e8e4799
# Created CodeCommit repository
# Generated buildspec file(s)
# Generated CloudFormation templates
# Committed files to CodeCommit repository
Pipeline resource template generation successful for application java-tomcat-9e8e4799

You're all set to use AWS CloudFormation to manage your pipeline stack.

Next Steps:
1. Edit the CloudFormation template as necessary.
2. Create a pipeline stack using the AWS CLI or the AWS Console. AWS CLI command:

aws cloudformation deploy --template-file /root/app2container/java-tomcat-9e8e4799/
Artifacts/Pipeline/CodePipeline/ecs-pipeline-master.yml --capabilities
 CAPABILITY_NAMED_IAM --stack-name app2container-java-tomcat-9e8e4799-ecs-pipeline-
stack
```

The following example shows the **generate pipeline** command with application id and the `--deploy` option.

```
$ sudo app2container generate pipeline --deploy --application id java-tomcat-9e8e4799
# Generated buildspec file(s)
# Generated CloudFormation templates
# Committed files to CodeCommit repository
# Initiated CloudFormation stack creation. This may take a few minutes. Please visit
 the AWS CloudFormation Console to track progress.
# Deployed pipeline through CloudFormation
Pipeline deployment successful for application --application id java-tomcat-9e8e4799

Successfully created AWS CodePipeline stack 'app2container---application id java-
tomcat-9e8e4799-ecs-pipeline-stack' for application. Check the AWS CloudFormation
 Console for additional details.
```

Windows

The following example shows the **generate pipeline** command with application id.

```
PS> app2container generate pipeline --application id iis-smarts-51d2dbf8
# Created CodeCommit repository
# Generated buildspec file(s)
# Generated CloudFormation templates
# Committed files to CodeCommit repository
Pipeline resource template generation successful for application --application id iis-
smarts-51d2dbf8

You're all set to use AWS CloudFormation to manage your pipeline stack.

Next Steps:
1. Edit the CloudFormation template as necessary.
2. Create a pipeline stack using the AWS CLI or the AWS Console. AWS CLI command:

aws cloudformation deploy --template-file C:\Users\Administrator\AppData\Local
\app2container\--application id iis-smarts-51d2dbf8\Artifacts\Pipeline\CodePipeline
\ecs-pipeline-master.yml --capabilities CAPABILITY_NAMED_IAM --stack-name
 app2container---application id iis-smarts-51d2dbf8-652becbe-ecs-pipeline-stack
```

The following example shows the **generate pipeline** command with application id and the `--deploy` option.

```
PS> app2container generate pipeline --deploy --application id iis-smarts-51d2dbf8
```

```
# Generated buildspec file(s)
# Generated CloudFormation templates
# Committed files to CodeCommit repository
# Initiated CloudFormation stack creation. This may take a few minutes. Please visit
 the AWS CloudFormation Console to track progress.
# Deployed pipeline through CloudFormation
Pipeline deployment successful for application --application id iis-smarts-51d2dbf8

Successfully created AWS CodePipeline stack 'app2container---application id iis-
smarts-51d2dbf8-ecs-pipeline-stack' for application. Check the AWS CloudFormation
 Console for additional details.
```

app2container help command

Lists the commands for App2Container, grouped into the phases where they would normally run.

Note
Commands are shown in alphabetical order within the phases where they run. For example, in the *Analyze* phase, you would run the **inventory** command first, then the **analyze** command.

Syntax

```
app2container help
```

Options

None

Output

The list of app2container commands.

Examples

```
app2container help
App2Container is an application from Amazon Web Services (AWS),
that provides commands to discover and containerize applications.

Commands
  Getting Started
    init         Sets up workspace for the app2container commands

  Analyze
    analyze      Analyzes selected application to identify dependencies required for
 containerization
    inventory    Lists all applications that can be containerized using app2container
 commands

  Transform
    containerize Generates Dockerfile, container images and deployment metadata
    extract      Creates an archive of application artifacts for containerization

  Deploy

  Settings
    upgrade      Upgrades app2container CLI to latest version
```

```
Flags
  -h, --help   help for app2container
```

app2container init command

The init command performs one-time initialization tasks for App2Container. Run this command before you run any other App2Container commands.

Syntax

```
app2container init [--advanced] [--help]
```

Options

--advanced

This option allows you to use features that are in the experimental phase, if any exist.

--help

Displays the command help.

Output

The **init** command prompts you for the information that it needs for initialization.

You must provide a local directory for application containerization artifacts. Ensure that only authorized users can access the local directory. If you do not specify a local directory, one is created for you at the default output location. The default locations are as follows:

- **Linux:** `/root/app2container`
- **Windows:** `C:\Users\Administrator\AppData\Local\app2container`

You can optionally provide an Amazon S3 bucket for application containerization artifacts. If you choose to set up an Amazon S3 bucket, you must ensure that only authorized users can access the bucket. We recommend that you use server-side encryption for your bucket. See Protecting data using server-side encryption in the *Amazon Simple Storage Service Developer Guide* for more information about how to set it up.

You can optionally consent to allow App2Container to collect and export the following metrics to AWS each time that you run an **app2container** command:

- Host OS name
- Host OS version
- Application stack type
- Application stack version
- JDK version (Linux only, for Java applications)
- App2Container CLI version
- Command that ran
- Command status

- Command duration
- Command features and flags
- Command errors
- Container base image

Examples

Choose the operating system tab for the application server or worker machine where you run the command.

Linux

The following example shows the **init** command with no additional options.

```
$ sudo app2container init
Please enter a workspace directory path to use for artifacts[default: /root/
app2container]:
Please enter an AWS Profile to use. (The same can be configured with 'aws configure --
profile <name>')[default: default]:
Please provide an S3 bucket to store application artifacts (Optional):
Please confirm permission to report usage metrics to AWS (Y/N)[default: y]:
Would you like to enforce the use of only signed images using Docker Content Trust
 (DCT)? (Y/N)[default: n]:
All application artifacts will be created under the above workspace. Please ensure that
 the folder permissions are secure.
Init configuration saved
```

The following example shows the **init** command with the `--advanced` option and default values.

```
PS> sudo app2container init --advanced
Please enter a workspace directory path to use for artifacts[default: /root/
app2container]:
Please enter an AWS Profile to use. (The same can be configured with 'aws configure --
profile <name>')[default: default]:
Please provide an S3 bucket to store application artifacts (Optional):
Please confirm permission to report usage metrics to AWS (Y/N)[default: y]:
Would you like to enforce the use of only signed images using Docker Content Trust
 (DCT)? (Y/N)[default: n]:
Would you like to enable experimental features? (Y/N)[default: n]:
All application artifacts will be created under the above workspace. Please ensure that
 the folder permissions are secure.
Init configuration saved
```

Windows

The following example shows the **init** command with no additional options.

```
PS> app2container init
Please enter a workspace directory path to use for artifacts[default: C:\Users
\Administrator\AppData\Local\app2container]:
Please enter an AWS Profile to use. (The same can be configured with 'aws configure --
profile <name>')[default: default]:
Please provide an S3 bucket to store application artifacts (Optional):
Please confirm permission to report usage metrics to AWS (Y/N)[default: y]:
Would you like to enforce the use of only signed images using Docker Content Trust
 (DCT)? (Y/N)[default: n]:
All application artifacts will be created under the above workspace. Please ensure that
 the folder permissions are secure.
```

```
Init configuration saved
```

The following example shows the **init** command with the `--advanced` option and default values.

```
PS> app2container init --advanced
Please enter a workspace directory path to use for artifacts[default: C:\Users
\Administrator\AppData\Local\app2container]:
Please enter an AWS Profile to use. (The same can be configured with 'aws configure --
profile <name>')[default: default]:
Please provide an S3 bucket to store application artifacts (Optional):
Please confirm permission to report usage metrics to AWS (Y/N)[default: y]:
Would you like to enforce the use of only signed images using Docker Content Trust
 (DCT)? (Y/N)[default: n]:
Please enter if we can enable checking for upgrades automatically (Y/N)[default: y]:
Would you like to enable experimental features? (Y/N)[default: n]:
All application artifacts will be created under the above workspace. Please ensure that
 the folder permissions are secure.
Init configuration saved
```

app2container inventory command

Displays all running Java processes (Linux) or all IIS websites (Windows) on the application server.

Syntax

```
app2container inventory [--help]
```

Options

--help

Displays the command help.

Output

Information about the Java processes or IIS websites is saved to the `inventory.json` file in one of the following locations:

- **Linux:** `/root/inventory.json`
- **Windows:** `C:\Users\Administrator\AppData\Local\.app2container-config\inventory.json`

The application ID that is used by other App2Container commands is the key for each application object in the JSON file. The application objects are slightly different depending on your application language and the application server operating system. Choose the operating system tab for your application in the Examples section below to see the differences.

Examples

Choose the operating system tab for the application server or worker machine where you run the command.

Linux

Each Java process has a unique application ID (for example, java-tomcat-9e8e4799). You can use this application ID with other AWS App2Container commands. Inventory information is saved to `/root/inventory.json`.

The following example shows the **inventory** command with no additional options.

```
$ sudo app2container inventory
{
    "java-jboss-5bbe0bec": {
        "processId": 27366,
        "cmdline": "java ...",
        "applicationType": "java-jboss"
    },
    "java-tomcat-9e8e4799": {
        "processId": 2537,
        "cmdline": "/usr/bin/java ...",
        "applicationType": "java-tomcat"
    }
}
```

Windows

Each IIS website has a unique application ID (for example, iis-smarts-51d2dbf8). You can use this application ID with other AWS App2Container commands. Inventory information is saved to C:\Users\Administrator\AppData\Local\.app2container-config\inventory.json.

The following example shows the **inventory** command with no additional options.

```
PS> app2container inventory
{
    "iis-smarts-51d2dbf8": {
        "siteName": "Default Web Site",
        "bindings": "http/*:80:,net.tcp/808:*",
        "applicationType": "iis",
        "discoveredWebApps": []
    },
    "iis-smart-544e2d61": {
        "siteName": "smart",
        "bindings": "http/*:82:",
        "applicationType": "iis",
        "discoveredWebApps": []
    }
}
```

app2container upgrade command

Run this command to upgrade your existing installation of App2Container.

If a newer version of AWS App2Container will break backwards compatibility with previously generated container artifacts when you do an upgrade, the upgrade command notifies you and requests permission to continue. If you choose to continue with the upgrade, you will be required to restart any ongoing analysis and containerization workflows for your applications.

Syntax

```
app2container upgrade [--help]
```

Options

--help

Displays the command help.

Output

Output is included in the Examples section for this command.

Examples

Choose the operating system tab for the application server or worker machine where you run the command.

Linux

Run the command shown below to upgrade your existing App2Container for Linux.

```
$ sudo app2container upgrade
Using version 1.0
Version 1.1 available for download
Starting Download...
Starting Installation...
Installation successful!
```

Windows

Run the command shown below to upgrade your existing App2Container for Windows.

```
PS> app2container upgrade
Using version 0.0
Version 2.0 available for download  Starting Download...
Starting Installation...Installation successful!
```

Troubleshooting App2Container issues

The following documentation can help you troubleshoot problems that you might have with the App2Container CLI.

Contents

Access App2Container logs on your server

A common first step in troubleshooting issues with any application is reviewing application logs. App2Container logs contain a history of the information and error messages that are produced by the commands that you run. If you opted out of metrics during initialization, the metrics messages are also logged in the local application log file.

Review log files in one of the following locations, depending on where you are running the command that needs troubleshooting:

Application logs

- **Linux:** `/root/app2container/log/app2container.log`
- **Windows:** `C:\Users\Administrator\AppData\Local\app2container\log\app2container.log`

Upgrade logs

- **Linux:** `/usr/local/app2container/log/app2container_upgrade.log`
- **Windows:** `C:\Users\Administrator\app2container\log\app2container_upgrade.log`

If there is more than one log file, it means that the first log file reached its maximum size, and a new log file was created to continue logging. Choose the most recent log file to troubleshoot.

Access application logs inside of a running container

You can access application logs on your running container by running a command shell from the container host that attaches to your container. Choose the tab that matches your container operating system to see the command.

Linux

From the host server, run an interactive bash shell on your running container.

```
$ docker exec -it container-id bash
```

Using the bash shell, you can then navigate to the location where your application logs are stored.

Windows

From the host server, run an interactive PowerShell session attached to your running container.

```
PS> docker exec -it container-id powershell.exe
```

Using the PowerShell session, you can then navigate to the location where your application logs are stored.

To look up Docker commands, use the Docker command line reference. See Use the Docker command line.

AWS resource creation fails for the generate command

Description

When you run the **generate app-deployment** or **generate pipeline** command, you receive an error message saying AWS resource creation has failed.

Cause

App2Container requires permission to access and create AWS resources when it generates and deploys application containers or pipelines. If the permission has not been configured in your IAM policy, or if you are using the default AWS profile for a command using the `--deploy` option, the command will fail.

Solution

Verify your IAM resources and AWS profile settings and adjust as necessary, depending on the command that failed and the details shown in your error message. For more information and instructions about how to set up IAM resources for App2Container, see Identity and access management in App2Container (p. 38).

Troubleshoot Java applications on Linux

This section contains issues you might have with using App2Container for Java applications running on Linux servers.

Application container image size is very large

Description

Your application container image is much larger than expected.

Cause

The application container image includes a kernel image with the application bits layered on top. The size of the image depends on both the size of the container operating system and the size of the application.

To catch all potential dependencies for Java applications on Linux that are not using JBoss or Tomcat frameworks, the container initially includes everything except the files that are already included in the kernel image.

Solution

Follow these steps to reduce the size of your application container image.

1. Use the `appExcludedFiles` section in your `analysis.json` file to exclude specific file and directory paths from the containerization process, and save the file when you are done.
2. Run the **containerize (p. 48)** command again to create a new application container image with the updates that you specified.

You can repeat this process as needed to further reduce the size.

Error: Insufficient disk space

Description

When you run the **containerize** command, it fails with the following error message: Error: Insufficient disk space.

Cause

For Java applications running on Linux, App2Container calculates the disk space that is required to generate the application container, and produces this error message if there is not enough free space. The calculation includes the space needed for the application archive (including all non-system files on the server), plus the space needed for **docker build** actions.

Solution

The error message generated by the **containerize** command includes the estimated space it needs to run successfully. There are many ways to address an insufficient space issue on Linux.

One way to ensure that your **containerize** command runs successfully is to reduce the size of the container that you are creating. Follow these steps to reduce the size of your application container image.

1. Use the `appExcludedFiles` section in your `analysis.json` file to exclude specific file and directory paths from the containerization process, and save the file when you are done.
2. Run the **containerize (p. 48)** command again to create a new application container image with the updates that you specified.

You can repeat this process as needed to further reduce the size.

Troubleshoot .NET applications on Windows

This section contains issues you might have with using App2Container for .NET applications running in IIS on Windows servers.

Application container image size is very large

Description

Your application container image is much larger than expected.

Cause

The application container image includes a kernel image with the application bits layered on top. The size of the image depends on both the size of the container operating system and the size of the application. The Windows Server Core image can be quite large, especially for versions prior to Windows Server Core 2019.

Solution

We recommend that you use Windows Server Core 2019 for your container operating system to create the smallest base container size possible.

Follow these steps to reduce the size of your application container image if you are not currently using Windows Server Core 2019 as your base image. To ensure that you get the correct version, specify the version tag as shown below. The repository for Windows base images does not support the concept of "latest" to target the most recent image version.

1. Use the `containerBaseImage` section in your `analysis.json` file to target the Windows Server Core 2019 base image tagged as `ltsc2019` and save the file when you are done.

   The `containerBaseImage` value includes both the image name and the `ltsc2019` tag, separated by a colon (:). For example: `"containerBaseImage": "mcr.microsoft.com/dotnet/framework/aspnet:4.7.2-windowsservercore-ltsc2019"`.
2. Run the **containerize (p. 48)** command again to create a new application container image. It will use the container operating system image that you specify in the `containerBaseImage` of your `analysis.json` file to build a new application container image.

Release notes for AWS App2Container

The following table describes the release history for AWS App2Container in descending date order:

Release date	Version	Details
September 15, 2020	1.0.2	• Added FireLens logging support • Added container image validation to pipeline generation • Bug fixes, including: ◦ Removed execution role in template if Windows is specified ◦ Fixed template to reflect CloudFormation API change ◦ Fixed autocomplete installation bug ◦ Fixed dark font for Windows errors ◦ Improved error messaging for command execution errors ◦ Fixed containerize error where

Release date	Version	Details
		included file is not valid

Release date	Version	Details
August 5, 2020	1.0.1	• Improved memory usage while archiving on Windows • Added support for containerizing individual applications running in Tomcat and JBoss standalone frameworks • Added schema version and unhealthy version checks • Bug fixes, including: ◦ Fixed handling for .NET Windows app running on alternative drives (not C) ◦ Fixed COPY command failure in DockerFile ◦ Access denied error now throws user error ◦ Added automatic removal of invalid characters from AppId ◦ Optimized Windows container image

Release date	Version	Details
		size for websites with multiple apps • Fixed error handling for input arguments validation • Fixed Dockerfile generation failure when dynamic logging is enabled • EKS CloudFormation templates are now compatible with the new CloudFormation custom resource API
June 30, 2020	1.0.0	• Initial release

Document history for AWS App2Container

The following table describes important changes to the documentation by date. For notification about updates to this documentation, you can subscribe to an RSS feed.

update-history-change	update-history-description	update-history-date
Release notes - v1.0.1 (p. 71)	Added Release notes page with version 1.0.1 changes for AWS App2Container.	August 5, 2020
Docs-only: configuration and IAM updates (p. 71)	A chapter was added to describe configurable fields in files generated by App2Container commands, and the security section was updated with an IAM best practices summary and guidance for setting up IAM general use resources for App2Container.	August 1, 2020
Initial release (p. 71)	This release introduces AWS App2Container.	June 30, 2020