User Guide

AWS OpsWorks



API Version 2013-02-18

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS OpsWorks: User Guide

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What Is OpsWorks? 1
OpsWorks Services 1
AWS OpsWorks for Puppet Enterprise 4
Region Support for OpsWorks for Puppet Enterprise 5
End of Life FAQs 6
How will existing customers be affected by this End of Life? 6
What happens to my servers if I don't take any action? 7
Is AWS OpsWorks for Puppet Enterprise accepting new customers? 7
Will the End of Life affect all AWS Regions at the same time? 7
What level of technical support is available for AWS OpsWorks for Puppet Enterprise? 7
I am a current customer of OpsWorks for Puppet Enterprise and I need to launch a server in an account which was not using the service previously. Am I able to do this? 7
Will there be any new feature releases for AWS OpsWorks for Puppet Enterprise? 8
Getting Started 8
Prerequisites 8
Create a Puppet Master 13
Finish Configuration 23
Add Nodes to Manage 27
Sign in to the Puppet Enterprise Console 30
Optional: Use CodeCommit 34
Create a Puppet Master in CloudFormation 41
Prerequisites 41
Create a Puppet Enterprise Master in AWS CloudFormation 42
Update a Server to Use a Custom Domain 47
Prerequisites 48
Limitations 48
Update a Server to Use a Custom Domain 49
See Also 52
Work with Tags 52
How Tags Work in AWS OpsWorks for Puppet Enterprise 54
Add and Manage Tags in OpsWorks for Puppet Enterprise (Console) 55
Add and Manage Tags in OpsWorks for Puppet Enterprise (CLI) 58
See Also 62
Back Up and Restore Servers 63

Back Up an OpsWorks for Puppet Enterprise Server ... 63
Restore an OpsWorks for Puppet Enterprise Server ... 67
System Maintenance ... 68
Configuring system maintenance ... 69
Starting system maintenance on demand ... 71
Restoring custom configurations and files after maintenance ... 72
Add Nodes Automatically ... 72
Step 1: Create an IAM Role to Use as Your Instance Profile ... 73
Step 2: Create Instances by Using an Unattended Association Script ... 74
Remove Nodes ... 75
See Also ... 76
Delete a Puppet Master ... 76
Step 1: Disassociate Managed Nodes ... 77
Step 2: Delete the Server ... 77
See Also ... 77
Migrate a Puppet Server to Amazon EC2 ... 78
Step 1: Contact Puppet to purchase a license ... 78
Step 2: Get details about your OpsWorks for Puppet Enterprise server ... 78
Step 3: Make a backup of your OpsWorks for Puppet Enterprise server ... 79
Step 4: Launch a new EC2 instance ... 80
Step 5: Install Puppet Enterprise on the new EC2 instance ... 81
Step 6: Restore the backup on the new EC2 instance ... 82
Step 7: Configure your Puppet license ... 82
Step 8: Migrate your nodes ... 82
Step 9: Delete your OpsWorks for Puppet Enterprise server ... 85
Using AWS CloudTrail ... 86
OpsWorks for Puppet Enterprise Information in CloudTrail ... 86
Understanding OpsWorks for Puppet Enterprise Log File Entries ... 87
Troubleshooting ... 89
General troubleshooting tips ... 90
Troubleshooting specific errors ... 90
Additional help and support ... 95
AWS OpsWorks for Chef Automate ... 96
Region Support for AWS OpsWorks for Chef Automate ... 99
End of Life FAQs ... 100
How will existing users be affected by this End of Life? ... 100

What happens to my servers if I don't take any action? ... 101
What alternatives can I transition to? ... 101
Is the service still accepting new customers? ... 101
Will the End of Life affect all AWS Regions at the same time? ... 101
What level of technical support is available? ... 101
I am a current customer of OpsWorks for Chef Automate and I need to launch a server in an account which was not using the service previously. Am I able to do this? ... 102
Will there be any major feature releases over the next year? ... 102
Upgrade to Chef Automate 2 ... 102
Prerequisites for Upgrading to Chef Automate 2 ... 102
About the Upgrade Process ... 103
Upgrade to Chef Automate 2 (Console) ... 103
Upgrade to Chef Automate 2 (CLI) ... 104
Roll Back an AWS OpsWorks for Chef Automate Server to Chef Automate 1 (CLI) ... 105
See Also ... 106
Getting Started ... 106
Prerequisites ... 107
Create a Chef Automate Server ... 109
Finish configuration and upload cookbooks ... 120
Add nodes to manage ... 129
Sign in to the Chef Automate dashboard ... 135
Create a Chef Automate Server in CloudFormation ... 138
Prerequisites ... 139
Create a Chef Automate Server in AWS CloudFormation ... 140
Update a Server to Use a Custom Domain ... 146
Prerequisites ... 146
Limitations ... 48
Update a Server to Use a Custom Domain ... 49
See Also ... 52
Regenerate the starter kit ... 151
Regenerate the AWS OpsWorks for Chef Automate starter kit with the AWS CLI ... 152
Work with Tags ... 153
How Tags Work in AWS OpsWorks for Chef Automate ... 154
Add and Manage Tags in AWS OpsWorks for Chef Automate (Console) ... 155
Add and Manage Tags in AWS OpsWorks for Chef Automate (CLI) ... 158
See Also ... 163

Back up and restore servers ... 163
Back Up an AWS OpsWorks for Chef Automate Server ... 164
Restore an AWS OpsWorks for Chef Automate Server ... 166
System Maintenance ... 168
Ensuring nodes trust the OpsWorks Certification Authority ... 169
Configuring system maintenance ... 170
Starting system maintenance on demand ... 172
Restoring custom configurations and files after maintenance ... 172
Compliance Scans ... 173
Compliance in Chef Automate 2.0 ... 174
Compliance in Chef Automate 1.*x* ... 182
Updates to Compliance ... 188
Community and Custom Compliance Profiles ... 188
See Also ... 188
Remove nodes ... 189
Related Topics ... 190
Delete a Chef Automate server ... 190
Step 1: Disassociate Managed Nodes ... 191
Step 2: Delete the Server ... 191
Reset Chef credentials ... 191
Using AWS CloudTrail ... 193
AWS OpsWorks for Chef Automate Information in CloudTrail ... 193
Understanding AWS OpsWorks for Chef Automate Log File Entries ... 194
Troubleshooting ... 196
General troubleshooting tips ... 197
Troubleshooting specific errors ... 197
Additional help and support ... 204
Security in AWS OpsWorks Configuration Management (CM) ... 206
Data Protection ... 207
Integration with AWS Secrets Manager ... 208
Data Encryption ... 209
Encryption at Rest ... 209
Encryption in Transit ... 209
Key Management ... 209
Identity and Access Management ... 209
Audience ... 210

Authenticating With Identities ... 210
Managing Access Using Policies ... 213
How OpsWorks CM Works with IAM ... 216
Identity-Based Policy Examples ... 221
Troubleshooting ... 224
AWS managed policies ... 226
Cross-service confused deputy prevention in AWS OpsWorks CM ... 234
Internetwork Traffic Privacy ... 238
Logging and Monitoring ... 239
Compliance Validation ... 239
Resilience ... 240
Infrastructure Security ... 240
Configuration and Vulnerability Analysis ... 241
Security Best Practices ... 241
OpsWorks Stacks ... 243
Stacks ... 246
Layers ... 246
Recipes and LifeCycle Events ... 246
Instances ... 247
Apps ... 248
Customizing your Stack ... 249
Resource Management ... 250
Security and Permissions ... 250
Monitoring and Logging ... 250
CLI, SDK, and AWS CloudFormation Templates ... 251
End of Life FAQs ... 251
How will existing customers be affected by this End of Life? ... 252
Is AWS OpsWorks Stacks accepting new customers? ... 252
Where should I migrate my existing stacks to? ... 252
How can I keep my existing Amazon EC2 instances after End of Life? ... 253
Will the End of Life affect all AWS Regions at the same time? ... 253
What level of technical support is available for AWS OpsWorks Stacks? ... 253
Will there be any new feature releases for AWS OpsWorks Stacks? ... 253
Migrating your applications to Systems Manager Application Manager ... 253
How the script works ... 254
Prerequisites ... 254

Limitations ... 255
Getting started ... 256
FAQ ... 271
Troubleshooting ... 283
Using the AWS OpsWorks Stacks Detach in Place tool ... 284
How the process works ... 285
Limitations ... 286
Getting started ... 287
Getting Started ... 296
Region Support ... 297
Getting Started: Sample ... 298
Getting Started: Linux ... 319
Getting Started: Windows ... 348
Getting Started: Cookbooks ... 381
Best Practices ... 415
Root Device Storage ... 416
Optimizing the Number of Servers ... 418
Managing Permissions ... 421
Managing and Deploying Apps and Cookbooks ... 424
Packaging Cookbook Dependencies Locally ... 433
Stacks ... 437
Migrate stacks from EC2-Classic ... 439
Create a New Stack ... 441
Running a Stack in a VPC ... 450
Update a Stack ... 461
Clone a Stack ... 462
Run Stack Commands ... 464
Using Custom JSON ... 467
Delete a Stack ... 470
Layers ... 474
OpsWorks Layer Basics ... 475
Elastic Load Balancing Layer ... 490
Amazon RDS Service Layer ... 495
ECS Cluster Layers ... 500
Custom Layers ... 507
Per-layer Package Installations ... 508

Instances ... 510
Using OpsWorks Stacks Instances ... 511
Using Computing Resources Created Outside of OpsWorks Stacks ... 570
Editing the Instance Configuration ... 617
Deleting OpsWorks Stacks Instances ... 619
Logging In with SSH ... 621
Logging In with RDP ... 624
Apps ... 628
Adding Apps ... 629
Deploying Apps ... 636
Editing Apps ... 640
Connecting to a Database ... 641
Using Environment Variables ... 643
Passing Data to Applications ... 645
Using Git Repository SSH Keys ... 648
Using Custom Domains ... 649
Using SSL ... 652
Cookbooks and Recipes ... 660
Cookbook Repositories ... 661
Chef Versions ... 664
Ruby Versions ... 682
Installing Custom Cookbooks ... 684
Updating Custom Cookbooks ... 687
Executing Recipes ... 690
Resource Management ... 698
Registering Resources with a Stack ... 700
Attaching and Moving Resources ... 705
Detaching Resources ... 711
Deregistering Resources ... 713
Tags ... 716
Setting Tags at the Stack Level ... 717
Setting Tags at the Layer Level ... 719
Managing Tags with the AWS CLI ... 721
Tag Limitations ... 722
Monitoring ... 723
Using Amazon CloudWatch ... 723

Using AWS CloudTrail ... 735
Using Amazon CloudWatch Logs ... 738
Using Amazon CloudWatch Events ... 743
Security and Permissions ... 744
Managing User Permissions ... 746
Allowing OpsWorks Stacks to Act on Your Behalf ... 768
Confused deputy prevention ... 773
Specifying Permissions for Apps Running on EC2 instances ... 776
Managing SSH Access ... 780
Managing Security Updates ... 787
Using Security Groups ... 788
Chef 12 Linux ... 792
Overview ... 792
Moving to Chef 12 ... 793
Supported Operating Systems ... 794
Supported Instance Types ... 794
More Information ... 795
Moving to Data Bags ... 795
Previous Chef Versions ... 797
Chef 11.10 and Earlier Versions for Linux ... 798
Using OpsWorks Stacks with Other AWS Services ... 1223
Using a Back-end Data Store ... 1224
ElastiCache Redis ... 1232
Using an Amazon S3 Bucket ... 1247
Using AWS CodePipeline with OpsWorks Stacks ... 1261
Using the OpsWorks Stacks CLI ... 1323
Create an Instance ... 1325
Deploy an App ... 1328
List Apps ... 1330
List Commands ... 1331
List Deployments ... 1332
List Elastic IP Addresses ... 1334
List Instances ... 1334
List Stacks ... 1336
List Layers ... 1338
Execute a Recipe ... 1342

Install Dependencies ... 1343
Update the Stack Configuration ... 1344
Debugging and Troubleshooting Guide ... 1344
Debugging Recipes ... 1346
Common Debugging and Troubleshooting Issues ... 1363
OpsWorks Stacks Agent CLI ... 1373
agent_report ... 1375
get_json ... 1376
instance_report ... 1380
list_commands ... 1381
run_command ... 1382
show_log ... 1383
stack_state ... 1384
OpsWorks Stacks Data Bag Reference ... 1387
App Data Bag (aws_opsworks_app) ... 1391
Command Data Bag (aws_opsworks_command) ... 1395
Amazon ECS Cluster Data Bag (aws_opsworks_ecs_cluster) ... 1397
Elastic Load Balancing Data Bag (aws_opsworks_elastic_load_balancer) ... 1398
Instance Data Bag (aws_opsworks_instance) ... 1399
Layer Data Bag (aws_opsworks_layer) ... 1404
Amazon RDS Data Bag (aws_opsworks_rds_db_instance) ... 1406
Stack Data Bag (aws_opsworks_stack) ... 1408
User Data Bag (aws_opsworks_user) ... 1410
OpsWorks Agent Changes ... 1411
Chef 12 Agent Releases ... 1411
Chef 11.10 Agent Releases ... 1415
Resources ... 1421
Reference Guides, Tools, and Support Resources ... 1421
AWS Software Development Kits ... 1422
Open Source Software ... 1423
OpsWorks document history ... 1424
Earlier updates ... 1431

What Is OpsWorks?

> **Important**
>
> The OpsWorks services have reached end of life and have been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks is a configuration management service that helps you configure and operate applications in a cloud enterprise by using Puppet or Chef. OpsWorks Stacks and AWS OpsWorks for Chef Automate let you use Chef cookbooks and solutions for configuration management, while OpsWorks for Puppet Enterprise lets you configure a Puppet Enterprise master server in AWS. Puppet offers a set of tools for enforcing the desired state of your infrastructure, and automating on-demand tasks.

OpsWorks Services

AWS OpsWorks for Puppet Enterprise

OpsWorks for Puppet Enterprise lets you create AWS-managed Puppet master servers. A Puppet master server manages nodes in your infrastructure, stores facts about those nodes, and serves as a central repository for your Puppet modules. Modules are reusable, shareable units of Puppet code that contain instructions about how your infrastructure should be configured. You can download community modules from the Puppet Forge, or use the Puppet Development Kit to create your own custom modules, then manage their deployment with Puppet Code Manager.

OpsWorks for Puppet Enterprise provides a fully-managed Puppet master, a suite of automation tools that enable you to inspect, deliver, operate, and future-proof your applications, and access to a user interface that lets you view information about your nodes and Puppet activities. OpsWorks for Puppet Enterprise lets you use Puppet to automate how nodes are configured, deployed, and managed, whether they are Amazon EC2 instances or on-premises devices. An OpsWorks for Puppet Enterprise master provides full-stack automation

by handling tasks such as software and operating system configurations, package installations, database setups, change management, policy enforcement, monitoring, and quality assurance.

Because OpsWorks for Puppet Enterprise manages Puppet Enterprise software, your server can be backed up automatically at a time that you choose, is always running the most current AWS-compatible version of Puppet, and always has the most current security updates applied. You can use Amazon EC2 Auto Scaling groups to associate new Amazon EC2 nodes with your server automatically.

AWS OpsWorks for Chef Automate

AWS OpsWorks for Chef Automate lets you create AWS-managed Chef servers that include Chef Automate premium features, and use the Chef DK and other Chef tooling to manage them. A Chef server manages nodes in your environment, stores information about those nodes, and serves as a central repository for your Chef cookbooks. The cookbooks contain recipes that are run by the Chef Infra client (`chef-client`) agent on each node that you manage by using Chef. You can use Chef tools like `knife` and Test Kitchen to manage nodes and cookbooks on a Chef server in the AWS OpsWorks for Chef Automate service.

Chef Automate is an included server software package that provides automated workflow for continuous deployment and compliance checks. AWS OpsWorks for Chef Automate installs and manages Chef Automate, Chef Infra, and Chef InSpec by using a single Amazon Elastic Compute Cloud instance. With AWS OpsWorks for Chef Automate, you can use community-authored or custom Chef cookbooks without making OpsWorks-specific changes.

Because AWS OpsWorks for Chef Automate manages Chef Automate components on a single instance, your server can be backed up automatically at a time that you choose, is always running the most current minor version of Chef, and always has the most current security updates applied. You can use Amazon EC2 Auto Scaling groups to associate new Amazon EC2 nodes with your server automatically.

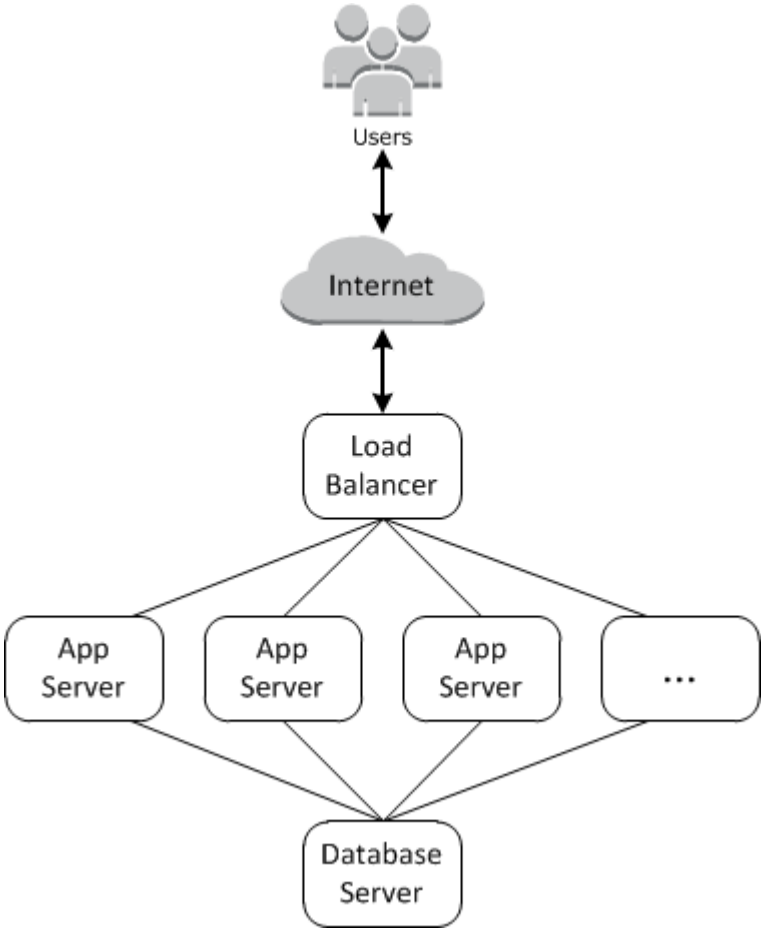
OpsWorks Stacks

Cloud-based computing usually involves groups of AWS resources, such as EC2 instances and Amazon Relational Database Service (RDS) instances. For example, a web application typically requires application servers, database servers, load balancers, and other resources. This group of instances is typically called a *stack*.

OpsWorks Stacks, the original service, provides a simple and flexible way to create and manage stacks and applications. OpsWorks Stacks lets you deploy and monitor applications in your

stacks. You can create stacks that help you manage cloud resources in specialized groups called *layers*. A layer represents a set of EC2 instances that serve a particular purpose, such as serving applications or hosting a database server. Layers depend on Chef recipes to handle tasks such as installing packages on instances, deploying apps, and running scripts.

Unlike AWS OpsWorks for Chef Automate, OpsWorks Stacks does not require or create Chef servers; OpsWorks Stacks performs some of the work of a Chef server for you. OpsWorks Stacks monitors instance health, and provisions new instances for you, when necessary, by using Auto Healing and Auto Scaling. A simple application server stack might look something like the following diagram.



AWS OpsWorks for Puppet Enterprise

Important

The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks for Puppet Enterprise lets you launch a Puppet Enterprise master in minutes, and lets OpsWorks handle its operations, backups, restorations, and software upgrades. OpsWorks for Puppet Enterprise frees you to focus on core configuration management tasks, instead of managing a Puppet master. By using OpsWorks for Puppet Enterprise, you can use the same configurations to manage both your on-premises and cloud infrastructure, helping you to efficiently scale your operations in a hybrid environment. Management of your Puppet master server is simplified by the Puppet Enterprise console, the AWS Management Console, and the AWS CLI.

A Puppet master manages the configuration of nodes in your environment by serving configuration catalogs for specific nodes to the `puppet-agent` software, and serves as a central repository for your Puppet modules. A Puppet master in OpsWorks for Puppet Enterprise deploys `puppet-agent` to your managed nodes, and provides premium features of Puppet Enterprise.

An OpsWorks for Puppet Enterprise master runs on an Amazon Elastic Compute Cloud instance. OpsWorks for Puppet Enterprise servers are configured to run the newest version of Amazon Linux (Amazon Linux 2), and the most current version of Puppet Enterprise Master, version 2019.8.5. For more information about changes in Puppet Enterprise 2019.8.5, see the Puppet Enterprise Release Notes.

When new versions of Puppet software become available, system maintenance is designed to update the version of Puppet Enterprise on the server automatically, as soon as it passes AWS testing. AWS performs extensive testing to verify that Puppet upgrades are production-ready and do not disrupt existing customer environments.

You can connect any on-premises computer or EC2 instance that is running a supported operating system and has network access to an OpsWorks for Puppet Enterprise master. The `puppet` agent software is installed by the Puppet master on nodes that you want to manage.

Topics

- Region Support for OpsWorks for Puppet Enterprise
- AWS OpsWorks for Puppet Enterprise End of Life FAQs
- Getting Started with OpsWorks for Puppet Enterprise
- Create an AWS OpsWorks for Puppet Enterprise Master by using AWS CloudFormation
- Update an OpsWorks for Puppet Enterprise Server to Use a Custom Domain
- Working with Tags on AWS OpsWorks for Puppet Enterprise Resources
- Back Up and Restore an OpsWorks for Puppet Enterprise Server
- System Maintenance in OpsWorks for Puppet Enterprise
- Adding Nodes Automatically in OpsWorks for Puppet Enterprise
- Disassociate a Node from an OpsWorks for Puppet Enterprise Server
- Delete an OpsWorks for Puppet Enterprise Server
- How to migrate an OpsWorks for Puppet Enterprise server to Amazon Elastic Compute Cloud (Amazon EC2)
- Logging OpsWorks for Puppet Enterprise API Calls with AWS CloudTrail
- Troubleshooting OpsWorks for Puppet Enterprise

Region Support for OpsWorks for Puppet Enterprise

The following regional endpoints support OpsWorks for Puppet Enterprise masters. OpsWorks for Puppet Enterprise creates resources that are associated with your Puppet masters, such as instance profiles, users, and service roles, in the same regional endpoint as your Puppet master. Your Puppet master must be in a VPC. You can use a VPC that you create or already have, or use the default VPC.

- US East (Ohio) Region
- US East (N. Virginia) Region
- US West (N. California) Region

- US West (Oregon) Region
- Asia Pacific (Tokyo) Region
- Asia Pacific (Singapore) Region
- Asia Pacific (Sydney) Region
- Europe (Frankfurt) Region
- Europe (Ireland) Region

AWS OpsWorks for Puppet Enterprise End of Life FAQs

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Topics

- How will existing customers be affected by this End of Life?
- What happens to my servers if I don't take any action?
- Is AWS OpsWorks for Puppet Enterprise accepting new customers?
- Will the End of Life affect all AWS Regions at the same time?
- What level of technical support is available for AWS OpsWorks for Puppet Enterprise?
- I am a current customer of OpsWorks for Puppet Enterprise and I need to launch a server in an account which was not using the service previously. Am I able to do this?
- Will there be any new feature releases for AWS OpsWorks for Puppet Enterprise?

How will existing customers be affected by this End of Life?

Existing customers will be unaffected until March 31, 2024, the End of Life date for OpsWorks for Puppet Enterprise. After the End of Life date, customers will no longer be able to manage their servers using the OpsWorks console or API.

What happens to my servers if I don't take any action?

Starting March 31, 2024, you will no longer be able to manage your servers using the OpsWorks console or API. At that time, we will stop performing any ongoing management functions for your servers such as backups or maintenance. To limit impact to customers, we will leave the EC2 instances running that back up Puppet Enterprise servers, but their licenses will no longer be valid as usage is no longer covered (or billed for) under the OpsWorks for Puppet Enterprise service agreement. If you want to continue managing your infrastructure with Puppet Enterprise, see How to migrate an OpsWorks for Puppet Enterprise server to Amazon Elastic Compute Cloud (Amazon EC2).

Is AWS OpsWorks for Puppet Enterprise accepting new customers?

No. AWS OpsWorks for Puppet Enterprise is no longer accepting new customers.

Will the End of Life affect all AWS Regions at the same time?

Yes. the API and Console will reach End of Life and be unusable as of March 31, 2024 in all Regions. For a list of AWS Regions where AWS OpsWorks for Puppet Enterprise is available, see AWS Regional Services List.

What level of technical support is available for AWS OpsWorks for Puppet Enterprise?

AWS will continue to provide the same level of support for AWS OpsWorks for Puppet Enterprise that customers have today up until the End of Life date. If you have questions or concerns, you can contact the AWS Support Team on AWS re:Post or through AWS Premium Support.

I am a current customer of OpsWorks for Puppet Enterprise and I need to launch a server in an account which was not using the service previously. Am I able to do this?

Generally not, unless there are exceptional circumstances to do so. If you have a special situation, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support with the details and justification for this and we will review your request.

Will there be any new feature releases for AWS OpsWorks for Puppet Enterprise?

No. As the service is reaching End of Life, we will not release any new features. However, we will continue to make security improvements and manage servers as expected until the End of Life date.

Getting Started with OpsWorks for Puppet Enterprise

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks for Puppet Enterprise lets you run a Puppet Enterprise server in AWS. You can provision a Puppet Enterprise master server in about 15 minutes.

Starting May 3, 2021, OpsWorks for Puppet Enterprise stores some Puppet Enterprise server attributes in AWS Secrets Manager. For more information, see Integration with AWS Secrets Manager.

The following walkthrough helps you create your first Puppet master in OpsWorks for Puppet Enterprise.

Prerequisites

Before you begin, you must complete the following prerequisites.

Topics

- Install the Puppet Development Kit
- Install the Puppet Enterprise Client Tools
- Set Up a Git Control Repository

- Set Up a VPC
- Set Up an EC2 Key Pair (Optional)
- Prerequisites for Using a Custom Domain (Optional)

Install the Puppet Development Kit

1. From the Puppet website, download the Puppet Development Kit that matches your local computer's operating system.
2. Install the Puppet Development Kit.
3. Add the Puppet Development Kit to your local computer's PATH variable.

   - On a Linux or macOS operating system, you can add the Puppet Development Kit to your PATH variable by running the following command in a Bash shell.

     ```
     echo 'export PATH=/opt/puppetlabs/pdk/bin/pdk:$PATH' >> ~/.bash_profile && source
      ~/.bash_profile
     ```

   - On a Windows-based operating system, you can add the Puppet Development Kit to your PATH variable by using the following .NET Framework command in a PowerShell session, or in the **Environment Variables** dialog box accessible from **System Properties**. You may need to run your PowerShell session as an administrator to run the following command.

     ```
     [Environment]::SetEnvironmentVariable("Path","new path value","Machine")
     ```

Install the Puppet Enterprise Client Tools

Puppet Enterprise (PE) client tools are a set of command-line tools that let you access Puppet Enterprise services from your workstation. The tools can be installed on many different operating systems, and they can also be installed on nodes that you are managing by using Puppet. For information about supported operating systems for the tools, and how to install them, see Installing PE client tools in the Puppet Enterprise documentation.

Set Up a Git Control Repository

Before you can launch a Puppet master, you must have a control repository configured in Git to store and change-manage your Puppet modules and classes. A URL to a Git repository and HTTPS or SSH account information to access the repository are required in the steps to launch your

Puppet Enterprise master server. For more information about how to set up a control repository that your Puppet Enterprise master will use, see Setting up a control repository. You can also find control repository setup instructions in the readme for Puppet's `control-repo` sample repository on GitHub. The structure of the control repository resembles the following.

```
### LICENSE
### Puppetfile
### README.md
### environment.conf
### hieradata
#   ### common.yaml
#   ### nodes
#       ### example-node.yaml
### manifests
#   ### site.pp
### scripts
#   ### code_manager_config_version.rb
#   ### config_version.rb
#   ### config_version.sh
### site
    ### profile
    #   ### manifests
    #       ### base.pp
    #       ### example.pp
    ### role
        ### manifests
            ### database_server.pp
            ### example.pp
            ### webserver.pp
```

Setting up a repository by using CodeCommit

You can create a new repository by using CodeCommit. For more information about how to use CodeCommit to create your control repository, see the section called “Optional: Use CodeCommit” in this guide. For more information about how to get started with Git on CodeCommit, see Getting started with AWS CodeCommit. To authorize your OpsWorks for Puppet Enterprise server for your repository, attach the `AWSCodeCommitReadOnly` policy to your IAM instance profile role.

Set Up a VPC

Your OpsWorks for Puppet Enterprise master must operate in an Amazon Virtual Private Cloud. You can add it to an existing VPC, use the default VPC, or create a new VPC to contain the server.

For information about Amazon VPC and how to create a new VPC, see the Amazon VPC Getting Started Guide.

If you create your own VPC, or use an existing one, the VPC should have the following settings or properties.

- The VPC should have at least one subnet.

  If your OpsWorks for Puppet Enterprise master will be publicly accessible, make the subnet public, and enable **Auto-assign public IP**.
- **DNS resolution** should be enabled.
- On the subnet, enable **Auto-assign public IP**.

Set Up an EC2 Key Pair (Optional)

An SSH connection is not necessary or recommended for typical management of the Puppet server; you can use the AWS Management Console and AWS CLI commands to perform many management tasks on your Puppet server.

An EC2 key pair is required to connect to your server by using SSH in the event that you lose or want to change the sign-in password for the Puppet Enterprise web-based console. You can use an existing key pair, or create a new key pair. For more information about how to create a new EC2 key pair, see Amazon EC2 Key Pairs.

If you don't need an EC2 key pair, you are ready to create a Puppet Enterprise master.

Prerequisites for Using a Custom Domain (Optional)

You can set up your Puppet Enterprise master on your own domain, specifying a public endpoint in a custom domain to use as the endpoint of your server. When you use a custom domain, all of the following are required, as described in detail in this section.

Topics

- Set Up a Custom Domain
- Get a Certificate
- Get a Private Key

Set Up a Custom Domain

To run your Puppet Enterprise master on your own custom domain, you will need a public endpoint of a server, such as `https://aws.my-company.com`. If you specify a custom domain, you must also provide a certificate and a private key, as described in the preceding sections.

To access the server after you create it, add a CNAME DNS record in your preferred DNS service. This record must point the custom domain to the endpoint (the value of the server's `Endpoint` attribute) that is generated by the Puppet master creation process. You cannot access the server by using the generated `Endpoint` value if the server is using a custom domain.

Get a Certificate

To set up your Puppet master on your own custom domain, you need A PEM-formatted HTTPS certificate. This can be be a single, self-signed certificate, or a certificate chain. As you complete the **Create a Puppet Enterprise Master** workflow, if you specify this certificate, you must also provide a custom domain and a private key.

The following are requirements for the certificate value:

- You can provide either a self-signed, custom certificate, or the full certificate chain.
- The certificate must be a valid X509 certificate, or a certificate chain in PEM format.
- The certificate must be valid at the time of upload. A certificate can't be used before its validity period begins (the certificate's `NotBefore` date), or after it expires (the certificate's `NotAfter` date).
- The certificate’s common name or subject alternative names (SANs), if present, must match the custom domain value.
- The certificate must match the value of the **Custom private key** field.

Get a Private Key

To set up your Puppet master on your own custom domain, you need a private key in PEM format for connecting to the server by using HTTPS. The private key must not be encrypted; it cannot be protected by a password or passphrase. If you specify a custom private key, you must also provide a custom domain and a certificate.

Create a Puppet Enterprise Master

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

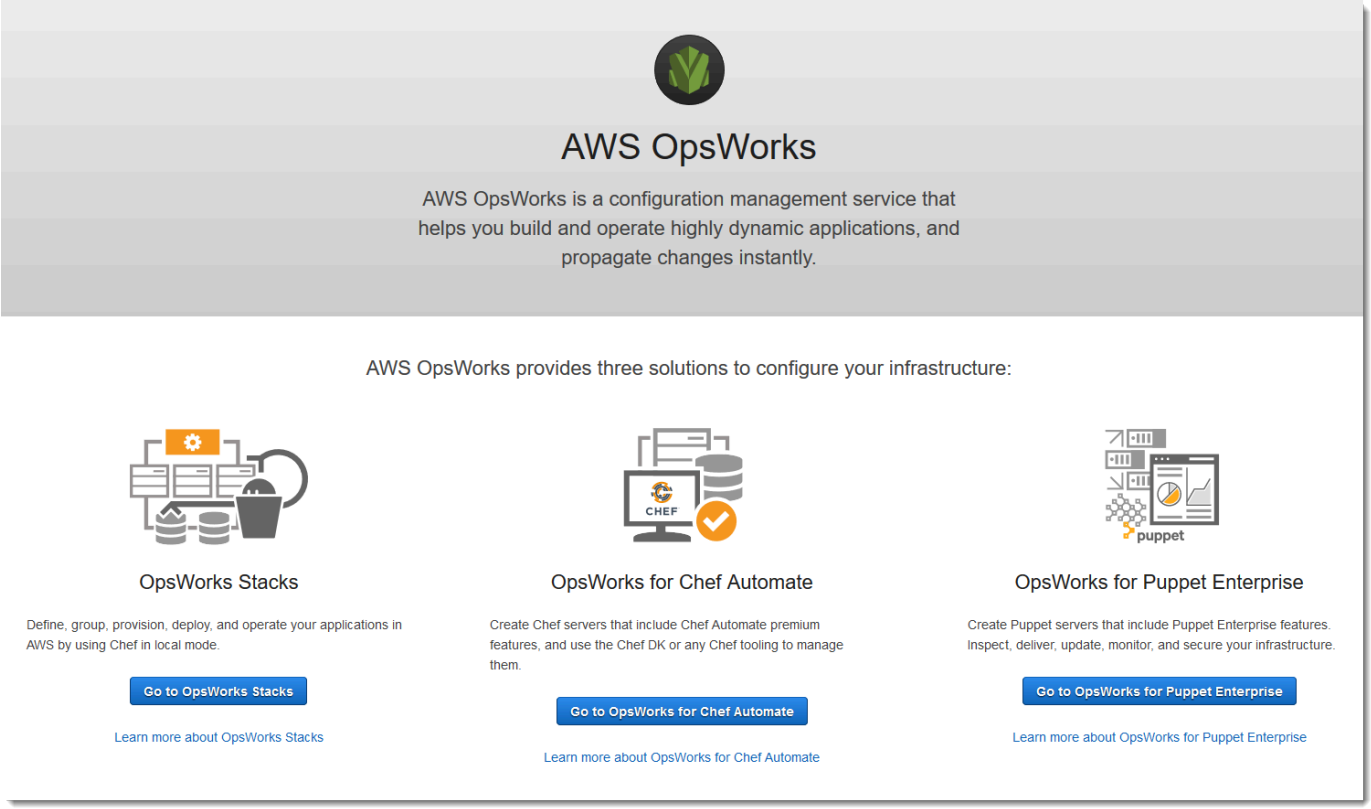
You can create a Puppet master by using the OpsWorks for Puppet Enterprise console, or the AWS CLI.

Topics

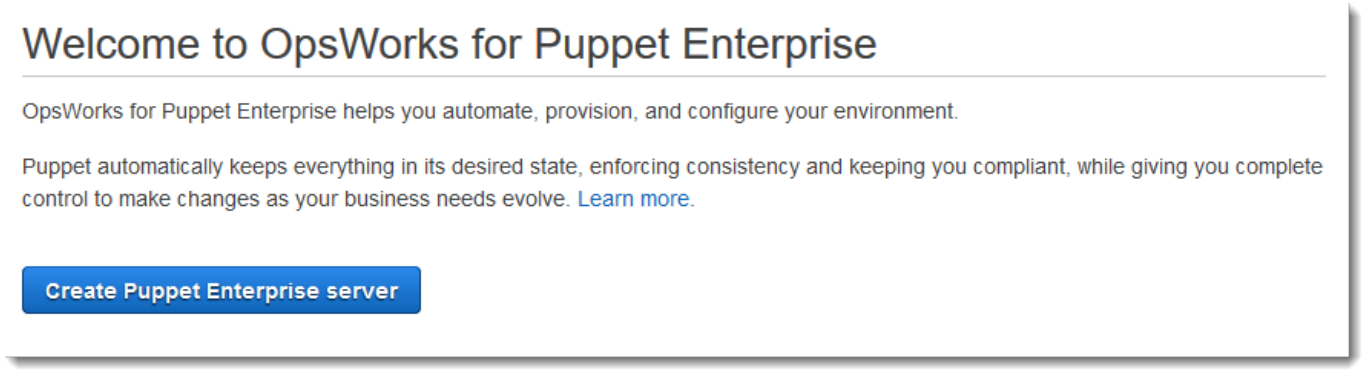
- Create a Puppet Enterprise Master by using the AWS Management Console
- Create a Puppet Enterprise Master by using the AWS CLI

Create a Puppet Enterprise Master by using the AWS Management Console

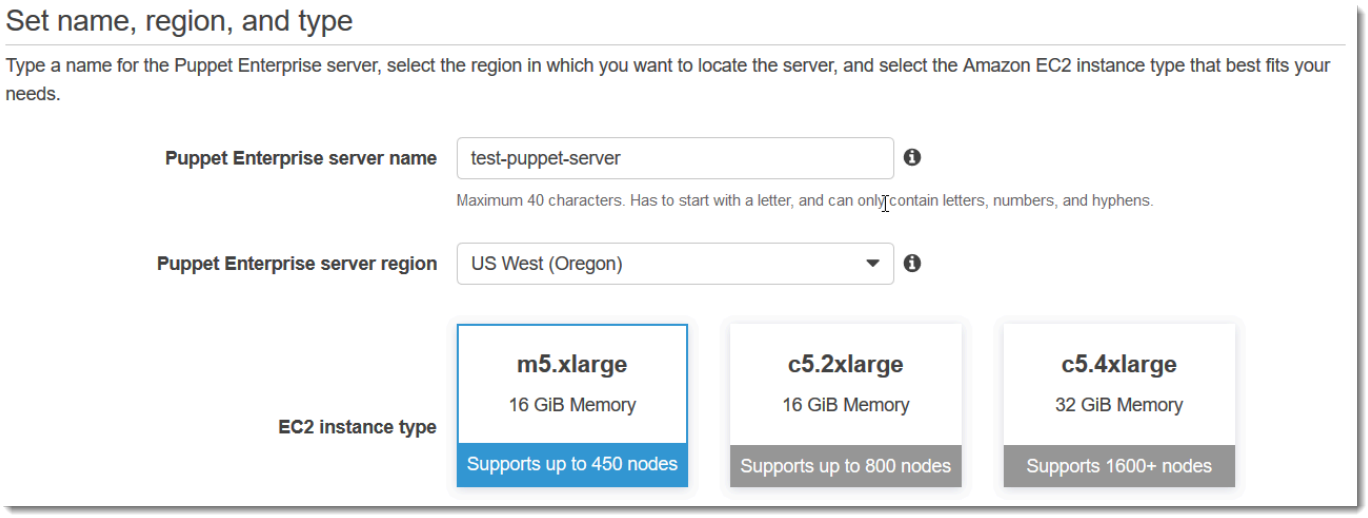
1. Sign in to the AWS Management Console and open the OpsWorks console at https://console.aws.amazon.com/opsworks/.
2. On the OpsWorks home page, choose **Go to OpsWorks for Puppet Enterprise**.



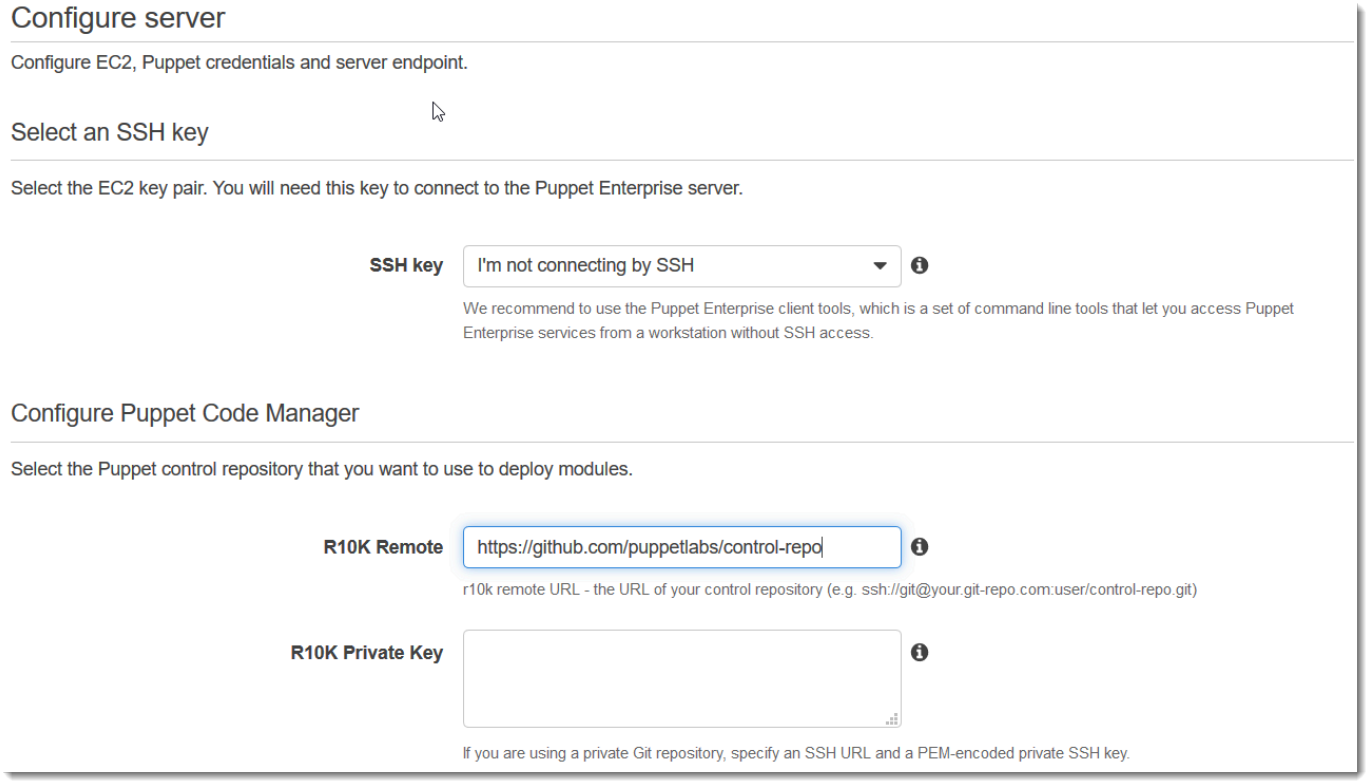
3. On the OpsWorks for Puppet Enterprise home page, choose **Create Puppet Enterprise server**.



4. On the **Set name, region, and type** page, specify a name for your server. Puppet master names can be a maximum of 40 characters, must start with a letter, and can contain only alphanumeric characters and dashes. Select a supported region, and then choose an instance type that supports the number of nodes that you want to manage. You can change the instance type after your server has been created, if needed. For this walkthrough, we are creating a **m5.xlarge** instance type in the US West (Oregon) Region. Choose **Next**.



5. On the **Configure server** page, leave the default selection in the **SSH key** drop-down list, unless you want to specify a key pair name. In the **r10k remote** field of the **Configure Puppet Code Manager** area, specify a valid SSH or HTTPS URL of your Git remote. In the **r10k private key** field, paste in the SSH private key that OpsWorks can use to access the r10k remote repository. This is provided by Git when you create a private repository, but not required if you are using HTTPS authentication to access your control repository. Choose **Next**.

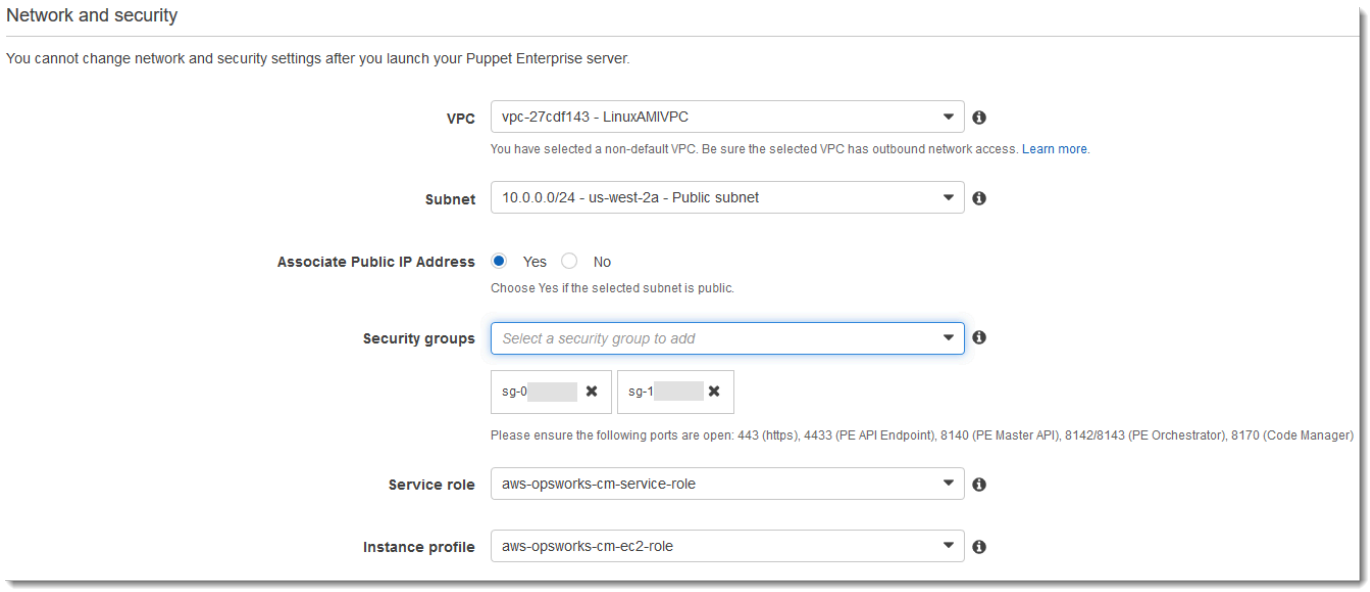


6. For **Specify server endpoint**, leave the default, **Use an automatically-generated endpoint** and then choose **Next**, unless you want your server on a custom domain of your own. To configure a custom domain, go on to the next step.
7. To use a custom domain, for **Specify server endpoint**, choose **Use a custom domain** from the drop-down list.

   a. For **Fully qualified domain name (FQDN)**, specify an FQDN. You must own the domain name that you want to use.

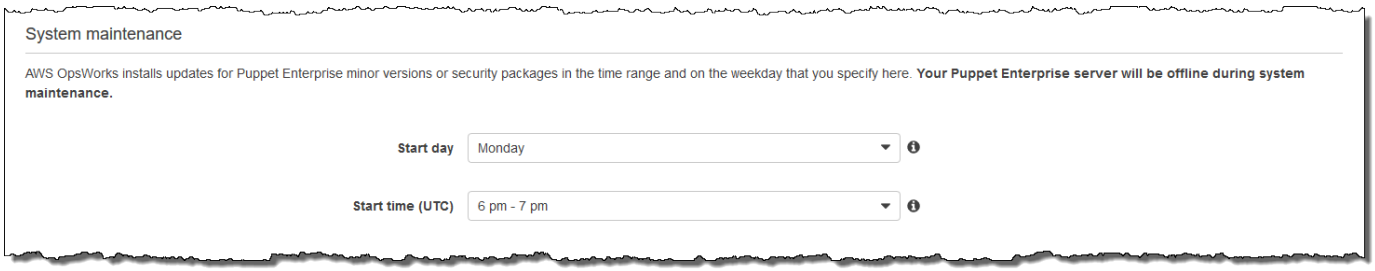
   b. For **SSL certificate**, paste in the entire PEM-formatted certificate, beginning with `-----BEGIN CERTIFICATE-----` and ending with `-----END CERTIFICATE-----`. The SSL certificate subject must match the FQDN you entered in the preceding step. Remove any extra lines before and after the certificate.

   c. For **SSL private key**, paste in the entire RSA private key, beginning with `-----BEGIN RSA PRIVATE KEY-----` and ending with `-----END RSA PRIVATE KEY-----`. The SSL private key must match the public key in the SSL certificate that you entered in the preceding step. Remove any extra lines before and after the private key. Choose **Next**.
8. On the **Configure advanced settings** page, in the **Network and security** area, choose a VPC, subnet, and one or more security groups. OpsWorks can generate a security group, service role, and instance profile for you, if you do not already have ones that you want to use. Your server can be a member of multiple security groups. You cannot change network and security settings for the Puppet master after you have left this page.

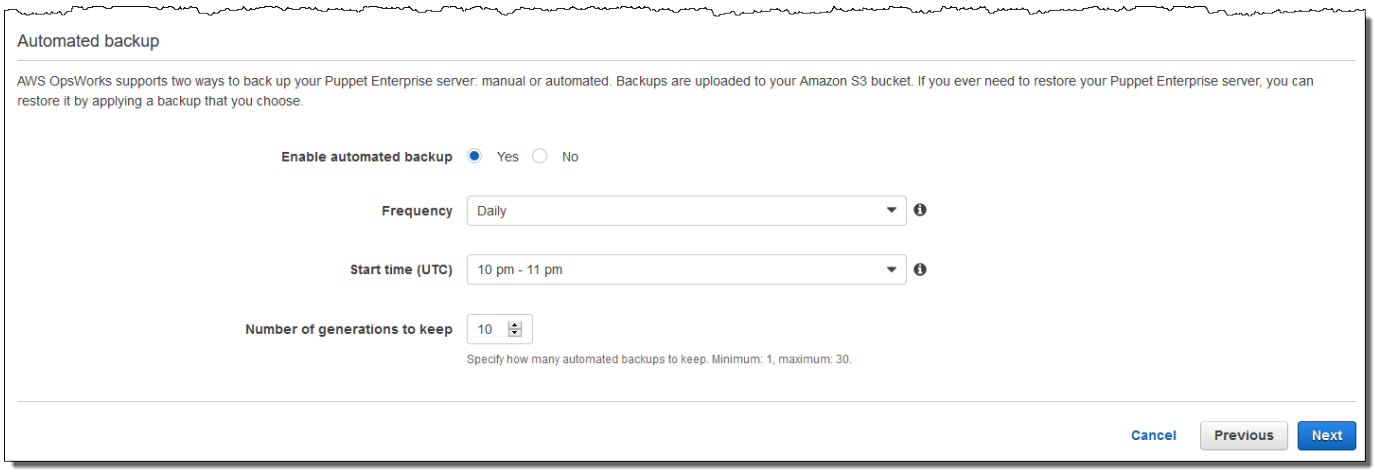


9. In the **System maintenance** section, set the day and hour that you want system maintenance to begin. Because you should expect the server to be offline during system maintenance, choose a time of low server demand within regular office hours.

   The maintenance window is required. You can change the start day and time later by using the AWS Management Console, AWS CLI, or the APIs.



10. Configure backups. By default, automatic backups are enabled. Set a preferred frequency and hour for automatic backup to start, and set the number of backup generations to store in Amazon Simple Storage Service. A maximum of 30 backups can be kept; when the maximum is reached, OpsWorks for Puppet Enterprise deletes the oldest backups to make room for new ones.



11. (Optional) In **Tags**, add tags to the server and related resources, such as the EC2 instance, Elastic IP address, security group, S3 bucket, and backups. For more information about tagging an OpsWorks for Puppet Enterprise server, see Working with Tags on AWS OpsWorks for Puppet Enterprise Resources.
12. When you are finished configuring advanced settings, choose **Next**.
13. On the **Review** page, review your choices. When you are ready to create the server, choose **Launch**.

While you are waiting for OpsWorks to create your Puppet master, go on to Configure the Puppet Master Using the Starter Kit and download the Starter Kit and the Puppet Enterprise console credentials. Do not wait until your server is online to download these items.

When server creation is finished, your Puppet master is available on the OpsWorks for Puppet Enterprise home page, with a status of **online**. After the server is online, the Puppet Enterprise console is available on the server's domain, at a URL in the following format: `https://your_server_name-randomID.region.opsworks-cm.io`.

Create a Puppet Enterprise Master by using the AWS CLI

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Creating an OpsWorks for Puppet Enterprise master server by running AWS CLI commands differs from creating a server in the console. In the console, OpsWorks creates a service role and security group for you, if you do not specify existing ones that you want to use. In the AWS CLI, OpsWorks can create a security group for you if you do not specify one, but it does not automatically create a service role; you must provide a service role ARN as part of your `create-server` command. In the console, while OpsWorks is creating your Puppet master, you download the starter kit and the sign-in credentials for the Puppet Enterprise console. Because you cannot do this when you create an OpsWorks for Puppet Enterprise master by using the AWS CLI, you use a JSON processing utility to get the sign-in credentials and the starter kit from the results of the `create-server` command after your new OpsWorks for Puppet Enterprise master is online.

If your local computer is not already running the AWS CLI, download and install the AWS CLI by following installation instructions in the *AWS Command Line Interface User Guide*. This section does not describe all parameters that you can use with the `create-server` command. For more information about `create-server` parameters, see `create-server` in the *AWS CLI Reference*.

1. Be sure to complete the Prerequisites. To create your Puppet master, you need a subnet ID, so you must have a VPC.
2. Create a service role and an instance profile.
3. Find and copy the ARNs of service roles in your account.

   ```
   aws iam list-roles --path-prefix "/service-role/" --no-paginate
   ```

   In the results of the `list-roles` command, look for service role ARN entries that resemble the following. Make a note of the service role ARNs. You need these values to create your Puppet Enterprise master.
4. Find and copy the ARNs of instance profiles in your account.

   ```
   aws iam list-instance-profiles --no-paginate
   ```

   In the results of the `list-instance-profiles` command, look for instance profile ARN entries that resemble the following. Make a note of the instance profile ARNs. You need these values to create your Puppet Enterprise master.
5. Create the OpsWorks for Puppet Enterprise master by running the `create-server` command.

   - The `--engine` value is `Puppet`, `--engine-model` is `Monolithic`, and `--engine-version` can be `2019` or `2017`.
   - The server name must be unique within your AWS account, within each region. Server names must start with a letter; then letters, numbers, or hyphens (-) are allowed, up to a maximum of 40 characters.
   - Use the instance profile ARN and service role ARN that you copied in Steps 3 and 4.
   - Valid instance types are `m5.xlarge`, `c5.2xlarge`, or `c5.4xlarge`. For more information about the specifications of these instance types, see Instance Types in the *Amazon EC2 User Guide*.
   - The `--engine-attributes` parameter is optional; if you don't specify a Puppet administrator password, the server creation process generates one for you. If you add `--engine-attributes`, specify a `PUPPET_ADMIN_PASSWORD`, an administrator password for signing in to the Puppet Enterprise console webpage. The password must use between 8 and 32 ASCII characters.

- An SSH key pair is optional, but can help you connect to your Puppet master if you need to reset the console administrator password. For more information about creating an SSH key pair, see Amazon EC2 Key Pairs in the *Amazon EC2 User Guide*.
- To use a custom domain, add the following parameters to your command. Otherwise, the Puppet master creation process automatically generates an endpoint for you. All three parameters are required to configure a custom domain. For information about additional requirements for using these parameters, see CreateServer in the OpsWorks CM API Reference.
  - `--custom-domain` - An optional public endpoint of a server, such as `https://aws.my-company.com`.
  - `--custom-certificate` - A PEM-formatted HTTPS certificate. The value can be be a single, self-signed certificate, or a certificate chain.
  - `--custom-private-key` - A private key in PEM format for connecting to the server by using HTTPS. The private key must not be encrypted; it cannot be protected by a password or passphrase.
- Weekly system maintenance is required. Valid values must be specified in the following format: `DDD:HH:MM`. The specified time is in coordinated universal time (UTC). If you do not specify a value for `--preferred-maintenance-window`, the default value is a random, one-hour period on Tuesday, Wednesday, or Friday.
- Valid values for `--preferred-backup-window` must be specified in one of the following formats: `HH:MM` for daily backups, or `DDD:HH:MM` for weekly backups. The specified time is in UTC. The default value is a random, daily start time. To opt out of automatic backups, add the parameter `--disable-automated-backup` instead.
- For `--security-group-ids`, enter one or more security group IDs, separated by a space.
- For `--subnet-ids`, enter a subnet ID.

```
aws opsworks-cm create-server --engine "Puppet" --engine-model "Monolithic"
 --engine-version "2019" --server-name "server_name" --instance-profile-arn
 "instance_profile_ARN" --instance-type "instance_type" --engine-attributes
 '{"PUPPET_ADMIN_PASSWORD":"ASCII_password"}' --key-pair "key_pair_name" --
preferred-maintenance-window "ddd:hh:mm" --preferred-backup-window "ddd:hh:mm"
 --security-group-ids security_group_id1 security_group_id2 --service-role-arn
 "service_role_ARN" --subnet-ids subnet_ID
```

The following is an example.

```
aws opsworks-cm create-server --engine "Puppet" --engine-model
 "Monolithic" --engine-version "2019" --server-name "puppet-02" --
instance-profile-arn "arn:aws:iam::111122223333:instance-profile/aws-
opsworks-cm-ec2-role" --instance-type "m5.xlarge" --engine-attributes
 '{"PUPPET_ADMIN_PASSWORD":"zZZzDj2DLYXSZFRv1d"}' --key-pair "amazon-test"
 --preferred-maintenance-window "Mon:08:00" --preferred-backup-window
 "Sun:02:00" --security-group-ids sg-b00000001 sg-b0000008 --service-role-arn
 "arn:aws:iam::111122223333:role/service-role/aws-opsworks-cm-service-role" --
subnet-ids subnet-383daa71
```

The following example creates a Puppet master that uses a custom domain.

```
aws opsworks-cm create-server \
    --engine "Puppet" \
    --engine-model "Monolithic" \
    --engine-version "2019" \
    --server-name "puppet-02" \
    --instance-profile-arn "arn:aws:iam::111122223333:instance-profile/aws-
opsworks-cm-ec2-role" \
    --instance-type "m5.xlarge" \
    --engine-attributes '{"PUPPET_ADMIN_PASSWORD":"zZZzDj2DLYXSZFRv1d"}' \
    --custom-domain "my-puppet-master.my-corp.com" \
    --custom-certificate "-----BEGIN CERTIFICATE----- EXAMPLEqEXAMPLE== -----END
 CERTIFICATE-----" \
    --custom-private-key "-----BEGIN RSA PRIVATE KEY----- EXAMPLEqEXAMPLE= -----END
 RSA PRIVATE KEY-----" \
    --key-pair "amazon-test"
    --preferred-maintenance-window "Mon:08:00" \
    --preferred-backup-window "Sun:02:00" \
    --security-group-ids sg-b00000001 sg-b0000008 \
    --service-role-arn "arn:aws:iam::111122223333:role/service-role/aws-opsworks-
cm-service-role" \
    --subnet-ids subnet-383daa71
```

The following example creates a Puppet master that adds two tags: `Stage: Production` and `Department: Marketing`. For more information about adding and managing tags on OpsWorks for Puppet Enterprise servers, see Working with Tags on AWS OpsWorks for Puppet Enterprise Resources in this guide.

```
aws opsworks-cm create-server \
```

```
    --engine "Puppet" \
    --engine-model "Monolithic" \
    --engine-version "2019" \
    --server-name "puppet-02" \
    --instance-profile-arn "arn:aws:iam::111122223333:instance-profile/aws-
opsworks-cm-ec2-role" \
    --instance-type "m5.xlarge" \
    --engine-attributes '{"PUPPET_ADMIN_PASSWORD":"zZZzDj2DLYXSZFRv1d"}' \
    --key-pair "amazon-test"
    --preferred-maintenance-window "Mon:08:00" \
    --preferred-backup-window "Sun:02:00" \
    --security-group-ids sg-b00000001 sg-b0000008 \
    --service-role-arn "arn:aws:iam::111122223333:role/service-role/aws-opsworks-
cm-service-role" \
    --subnet-ids subnet-383daa71 \
    --tags [{\"Key\":\"Stage\",\"Value\":\"Production\"},{\"Key\":\"Department\",
\"Value\":\"Marketing\"}]
```

6. OpsWorks for Puppet Enterprise takes about 15 minutes to create a new server. Do not dismiss the output of the `create-server` command or close your shell session, because the output can contain important information that is not shown again. To get passwords and the starter kit from the `create-server` results, go on to the next step.

   If you are using a custom domain with the server, in the output of the `create-server` command, copy the value of the `Endpoint` attribute. The following is an example.

   ```
   "Endpoint": "puppet-07-exampleexample.opsworks-cm.us-east-1.amazonaws.com"
   ```

7. If you opted to have OpsWorks for Puppet Enterprise generate a password for you, you can extract it in a usable format from the `create-server` results by using a JSON processor such as jq. After you install jq, you can run the following commands to extract the Puppet administrator password and starter kit. If you did not provide your own password in Step 3, be sure to save the extracted administrator password in a convenient but secure location.

   ```
   #Get the Puppet password:
   cat resp.json | jq -r '.Server.EngineAttributes[] | select(.Name ==
    "PUPPET_ADMIN_PASSWORD") | .Value'

   #Get the Puppet Starter Kit:
   cat resp.json | jq -r '.Server.EngineAttributes[] | select(.Name ==
    "PUPPET_STARTER_KIT") | .Value' | base64 -D > starterkit.zip
   ```

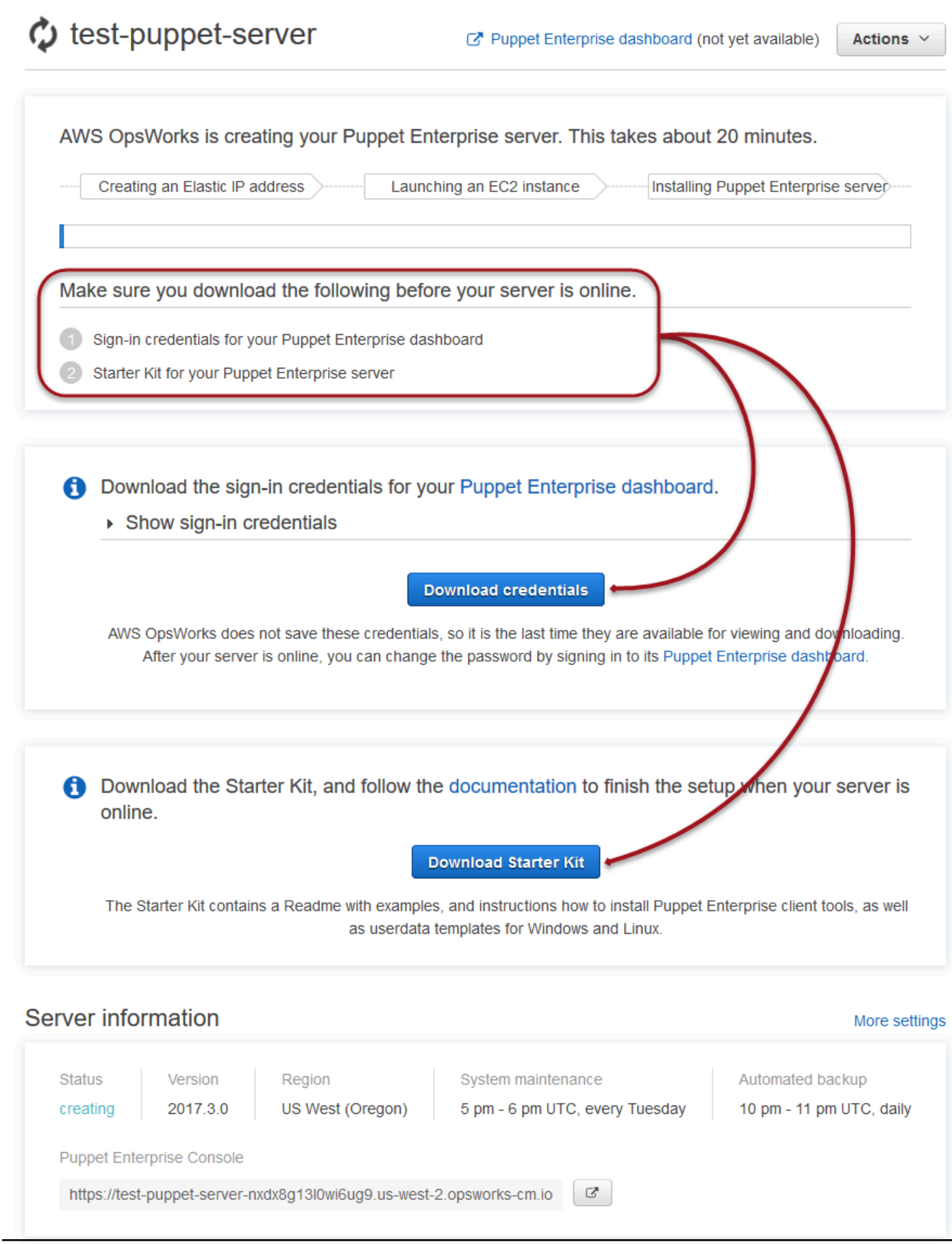
> **Note**
>
> You cannot regenerate a new Puppet master starter kit in the AWS Management Console. When you create a Puppet master by using the AWS CLI, run the preceding `jq` command to save the base64-encoded starter kit in the `create-server` results as a ZIP file.

8. If you are not using a custom domain, go on to the next step. If you are using a custom domain with the server, create a CNAME entry in your enterprise's DNS management tool to point your custom domain to the OpsWorks for Puppet Enterprise endpoint that you copied in step 6. You cannot reach or sign in to a server with a custom domain until you complete this step.
9. Go on to the next section, the section called “Finish Configuration”.

Configure the Puppet Master Using the Starter Kit

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

While Puppet master creation is still in progress, the server's Properties page opens in the OpsWorks for Puppet Enterprise console. The first time that you work with a new Puppet master, the Properties page prompts you to download two required items. Download these items before your Puppet server is online; the download buttons are not available after a new server is online.


test-puppet-server
Puppet Enterprise dashboard (not yet available)
Actions
AWS OpsWorks is creating your Puppet Enterprise server. This takes about 20 minutes.
Creating an Elastic IP address
Launching an EC2 instance
Installing Puppet Enterprise server
Make sure you download the following before your server is online.
1 Sign-in credentials for your Puppet Enterprise dashboard
2 Starter Kit for your Puppet Enterprise server
Download the sign-in credentials for your Puppet Enterprise dashboard.
Show sign-in credentials
Download credentials
AWS OpsWorks does not save these credentials, so it is the last time they are available for viewing and downloading.
After your server is online, you can change the password by signing in to its Puppet Enterprise dashboard.
Download the Starter Kit, and follow the documentation to finish the setup when your server is online.
Download Starter Kit
The Starter Kit contains a Readme with examples, and instructions how to install Puppet Enterprise client tools, as well as userdata templates for Windows and Linux.
Server information
More settings
Status
creating
Version
2017.3.0
Region
US West (Oregon)
System maintenance
5 pm - 6 pm UTC, every Tuesday
Automated backup
10 pm - 11 pm UTC, daily
Puppet Enterprise Console
https://test-puppet-server-nxdx8g13l0wi6ug9.us-west-2.opsworks-cm.io

- **Sign-in credentials for the Puppet master.** You will use these credentials to sign in to the Puppet Enterprise console, where you perform most node management. OpsWorks does not save these credentials; this is the last time that they are available for viewing and downloading. If necessary, you can change the password that is provided with these credentials after you sign in.
- **Starter Kit.** The Starter Kit contains a README file with information and examples describing how to finish setup, and administrator credentials for the Puppet Enterprise console. New credentials are generated—and the old credentials invalidated—each time you download the Starter Kit.

Prerequisites

1. While server creation is still in progress, download the sign-in credentials for the Puppet master, and save them in a secure but convenient location.
2. Download the Starter Kit, and unzip the Starter Kit .zip file into your workspace directory. Do not share your sign-in credentials. If other users will be managing the Puppet master, add them as administrators in the Puppet Enterprise console later. For more information about how to add users to the Puppet master, see Creating and managing users and user roles in the Puppet Enterprise documentation.

Install the Puppet Master Certificate

To work with your Puppet master and add nodes to manage, you'll need to install its certificate. Install it by running the following AWS CLI command. You cannot perform this task in the AWS Management Console.

```
aws --region region opsworks-cm describe-servers --server-name server_name --query
 "Servers[0].EngineAttributes[?Name=='PUPPET_API_CA_CERT'].Value" --output text
 > .config/ssl/cert/ca.pem
```

Generate a Short-term Token

To use the Puppet API, you must create a short-term token for yourself. This step is not required to use the Puppet Enterprise console. Generate the token by running the following command.

The default token lifetime is five minutes, but you can change this default.

```
puppet-access login --config-file .config/puppetlabs/client-tools/puppet-access.conf --
lifetime 8h
```

> **Note**
>
> Because the default token lifetime is five minutes, the preceding example command adds the `--lifetime` parameter to extend the token lifetime for a longer period. You can set the token lifetime for a period of up to 10 years (`10y`). For more information about how to change the default token lifetime, see Change the token's default lifetime in the Puppet Enterprise documentation.

Set Up the Starter Kit Apache Example

After you download and unzip the Starter Kit, you can use the example branch in the included, sample `control-repo-example` folder to configure an Apache web server on your managed nodes.

The Starter Kit includes two `control-repo` folders: `control-repo`, and `control-repo-example`. The `control-repo` folder includes a `production` branch that is unchanged from what you would see in the Puppet GitHub repository. The `control-repo-example` folder also has a `production` branch that includes example code to set up an Apache server with a test website.

1. Push the `control-repo-example` `production` branch to your Git remote (the `r10k_remote` URL of your Puppet master). In your Starter Kit root directory, run the following, replacing *`r10kRemoteUrl`* with your `r10k_remote` URL.

   ```
   cd  control-repo-example
   git remote add origin r10kRemoteUrl
   git push origin production
   ```

   Puppet's Code Manager uses Git branches as environments. By default, all nodes are in the production environment.

> **Important**
>
> Do not push to a `master` branch. The `master` branch is reserved for the Puppet master.

2. Deploy the code in the `control-repo-example` branch to your Puppet master. This lets the Puppet Master download your Puppet code from your Git repository (`r10k_remote`). In your Starter Kit root directory, run the following.

```
puppet-code deploy --all --wait --config-file .config/puppet-code.conf
```

For more information about how you can apply the sample Apache configuration to managed nodes that you create in Amazon EC2, see Step 2: Create Instances by Using an Unattended Association Script in this guide.

Add Nodes for the Puppet Master to Manage

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Topics

- Run associateNode() API calls
- Considerations for Adding On-premises Nodes
- More Information

The recommended way to add nodes is by using the OpsWorks `associateNode()` API. The Puppet Enterprise master server hosts a repository that you use to install the Puppet agent software on nodes that you want to manage, whether nodes are on-premises physical computers or virtual machines. Puppet agent software for some operating systems is installed on the

OpsWorks for Puppet Enterprise server as part of the launch process. The following table shows the operating system agents that are available on your OpsWorks for Puppet Enterprise server at launch.

Preinstalled operating system agents

Supported operating system	Versions
Ubuntu	16.04, 18.04, 20.04
Red Hat Enterprise Linux (RHEL)	6, 7, 8
Windows	64-bit editions of all Puppet-supported Windows releases

You can add `puppet-agent` to your server for other operating systems. Be aware that system maintenance will delete agents that you have added to your server after launch. Although most existing attached nodes that are already running the deleted agent continue to check in, nodes running Debian operating systems can stop reporting. We recommend that you manually install `puppet-agent` on nodes that are running operating systems for which the agent software is not preinstalled on your OpsWorks for Puppet Enterprise server. For detailed information about how to make `puppet-agent` available on your server for nodes with other operating systems, see Installing agents in the Puppet Enterprise documentation.

For information about how to associate nodes with your Puppet master automatically by populating EC2 instance user data, see Adding Nodes Automatically in OpsWorks for Puppet Enterprise.

Run `associateNode()` API calls

After you add nodes by installing `puppet-agent`, nodes send certificate signing requests (CSRs) to the OpsWorks for Puppet Enterprise server. You can view the CSRs in the Puppet console; for more information about node CSRs, see Managing certificate signing requests in the Puppet Enterprise documentation. Running the OpsWorks for Puppet Enterprise `associateNode()` API call processes node CSRs, and associates the node with your server. The following is an example of how to use this API call in the AWS CLI to associate a single node. You will need the PEM-formatted CSR that the node sends; you can get this from the Puppet console.

```
aws opsworks-cm associate-node --server-name "test-puppet-
server" --node-name "node or instance ID" --engine-attributes
 "Name=PUPPET_NODE_CSR,Value='PEM_formatted_CSR_from_the_node'
```

For more information about how to add nodes automatically by using `associateNode()`, see Adding Nodes Automatically in OpsWorks for Puppet Enterprise.

Considerations for Adding On-premises Nodes

After you have installed `puppet-agent` on your on-premises computers or virtual machines, you can use either of two ways to associate on-premises nodes with your OpsWorks for Puppet Enterprise master.

- If a node supports installation of the AWS SDK, AWS CLI, or AWS Tools for PowerShell, you can use the recommended method for associating a node, which is to run an `associateNode()` API call. The starter kit that you download when you first create an OpsWorks for Puppet Enterprise master shows how to assign roles to nodes by using tags. You can apply tags at the same time that you are associating nodes with the Puppet master by specifying trusted facts in the CSR. For example, the demo control repository that is included with the starter kit is configured to use the tag `pp_role` to assign roles to Amazon EC2 instances. For more information about how to add tags to a CSR as trusted facts, see Extension requests (permanent certificate data) in the Puppet platform documentation.
- If the node cannot run AWS management or development tools, you can still register it with your OpsWorks for Puppet Enterprise master the same way you would register it with any unmanaged Puppet Enterprise master. As mentioned in this topic, installing `puppet-agent` sends a CSR to the OpsWorks for Puppet Enterprise master. An authorized Puppet user can sign the CSR manually, or configure automatic signing of CSRs by editing the `autosign.conf` file that is stored on the Puppet master. For more information about configuring autosigning and editing `autosign.conf`, see SSL configuration: autosigning certificate requests in the Puppet platform documentation.

To associate on-premises nodes with a Puppet master and allow the Puppet master to accept all CSRs, do the following in the Puppet Enterprise console. The parameter that controls the behavior is `puppet_enterprise::profile::master::allow_unauthenticated_ca`.

> **Important**
>
> Enabling the Puppet master to accept self-signed CSRs or all CSRs is not recommended for security reasons. By default, allowing unauthenticated CSRs makes a Puppet master accessible to the world. Setting the upload of certificate requests to be enabled by default can make your Puppet master vulnerable to denial of service (DoS) attacks.

1. Sign in to the Puppet Enterprise console.
2. Choose **Configure**, choose **Classification**, choose **PE Master**, and then choose the **Configuration** tab.
3. On the **Classification** tab, locate the class **puppet_enterprise::profile::master**.
4. Set the value of the **allow_unauthenticated_ca** parameter to **true**.
5. Save your changes. Your changes are applied during the next Puppet run. You can allow 30 minutes for changes to take effect (and on-premises nodes to be added), or you can initiate a Puppet run manually in the **Run** section of the PE console.

More Information

Visit the Learn Puppet tutorial site to learn more about using OpsWorks for Puppet Enterprise servers and Puppet Enterprise console features.

Sign in to the Puppet Enterprise Console

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

After you have downloaded the sign-in credentials from the Puppet master's Properties page, and the server is online, sign in to the Puppet Enterprise console. In this walkthrough, we instructed you to specify your control repository that contains your modules, and add at least one node to manage. This allows you to see information about the agent and nodes in the console.

When you attempt to connect to the Puppet Enterprise console webpage, certificate warnings appear in your browser until you install an OpsWorks-specific, CA-signed SSL certificate on the client computer that you are using to manage your Puppet server. If you prefer not to see the warnings before you continue to the dashboard webpage, install the SSL certificate before you sign in.

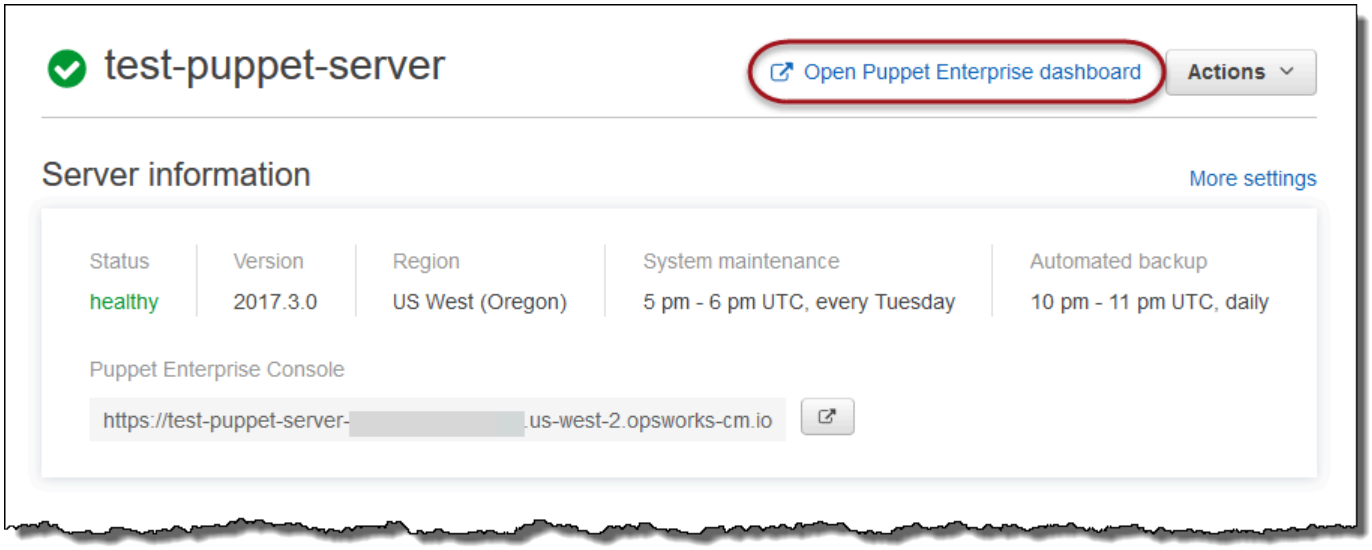
To install the OpsWorks SSL certificate

- Choose the certificate that matches your system.

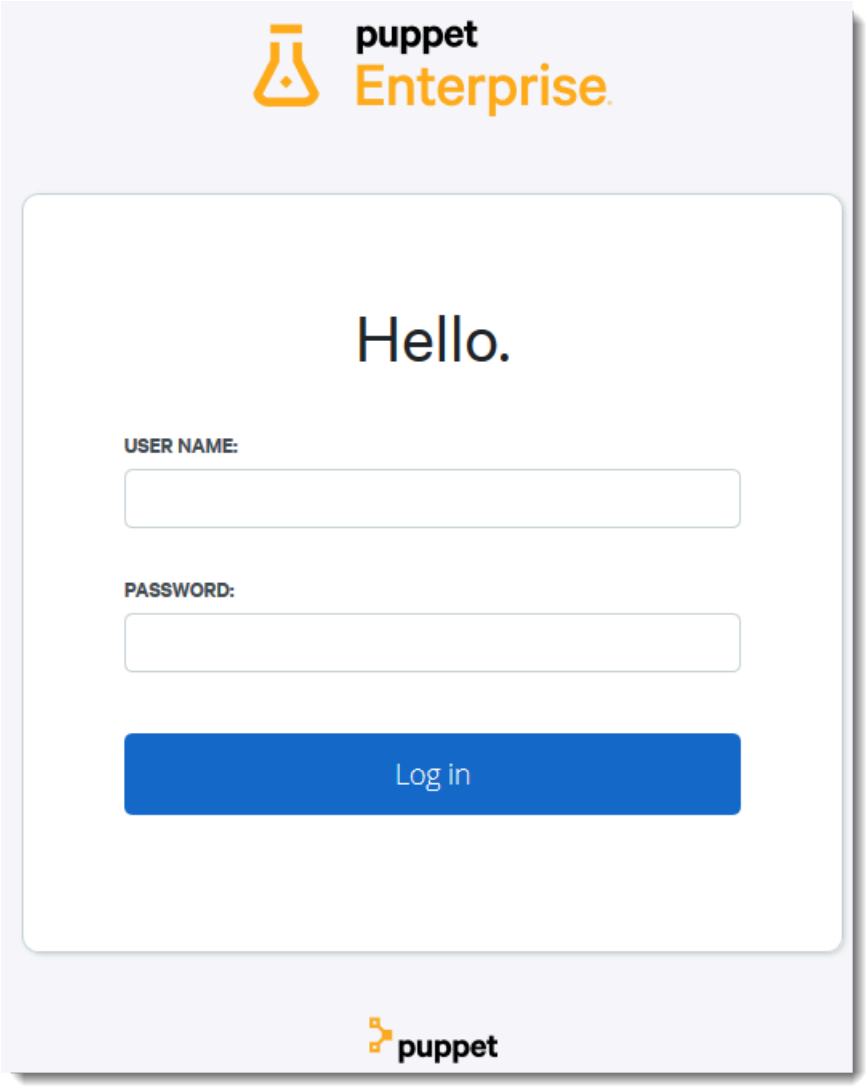
After you have installed the client-side SSL certificate, you can sign in to the Puppet Enterprise console without seeing warning messages.

To sign in to the Puppet Enterprise console

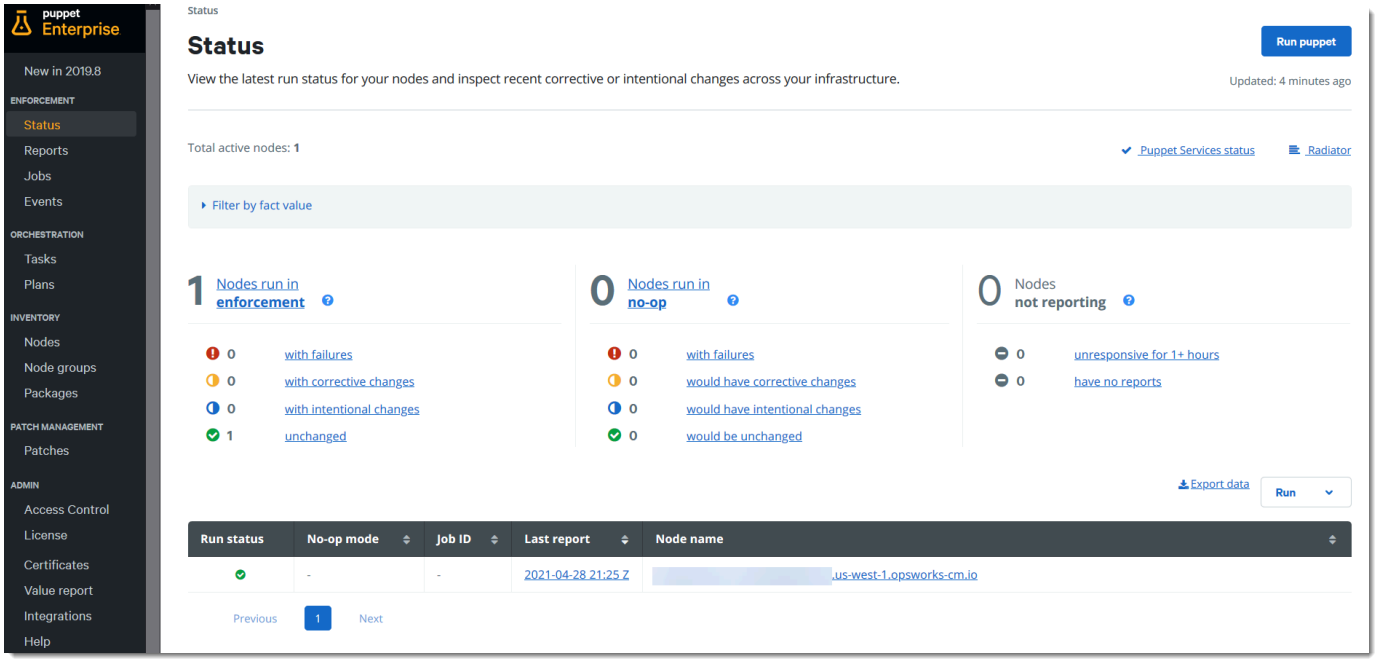
1. Unzip and open the Puppet Enterprise credentials that you downloaded in Prerequisites. You will need these credentials to sign in.
2. In the AWS Management Console, open the **Properties** page for your Puppet server.
3. At the upper right of the **Properties** page, choose **Open Puppet Enterprise console**.



4. Sign in using the credentials from Step 1.



5. In the Puppet Enterprise console, you can view detailed information about the nodes you're managing, module run progress and events, the compliance level of nodes, and much more. For more information about the features of the Puppet Enterprise console and how to use them, see Managing nodes in the Puppet Enterprise documentation.



Group and Classify Nodes

Before you specify the desired configuration of your nodes by applying classes to them, group the nodes according to their roles in your enterprise or their common characteristics. Grouping and classifying nodes involves the following high-level tasks. You can complete these tasks by using the PE console. For detailed information about how to group and classify your nodes, see Grouping and classifying nodes in the Puppet Enterprise documentation.

1. Create node groups.
2. Add nodes to groups manually or automatically by applying rules that you create.
3. Assign classes to node groups.

Reset Administrator and User Passwords

For information about how to change the password that you use to sign in to the Puppet Enterprise console, see Reset the console administrator password in the Puppet Enterprise documentation.

By default, after ten sign-in attempts, users are locked out of the Puppet console. For more information about how to reset user passwords in the event of a lockout, see Password endpoints in the Puppet Enterprise documentation.

Optional: Use AWS CodeCommit as a Puppet r10k Remote Control Repository

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

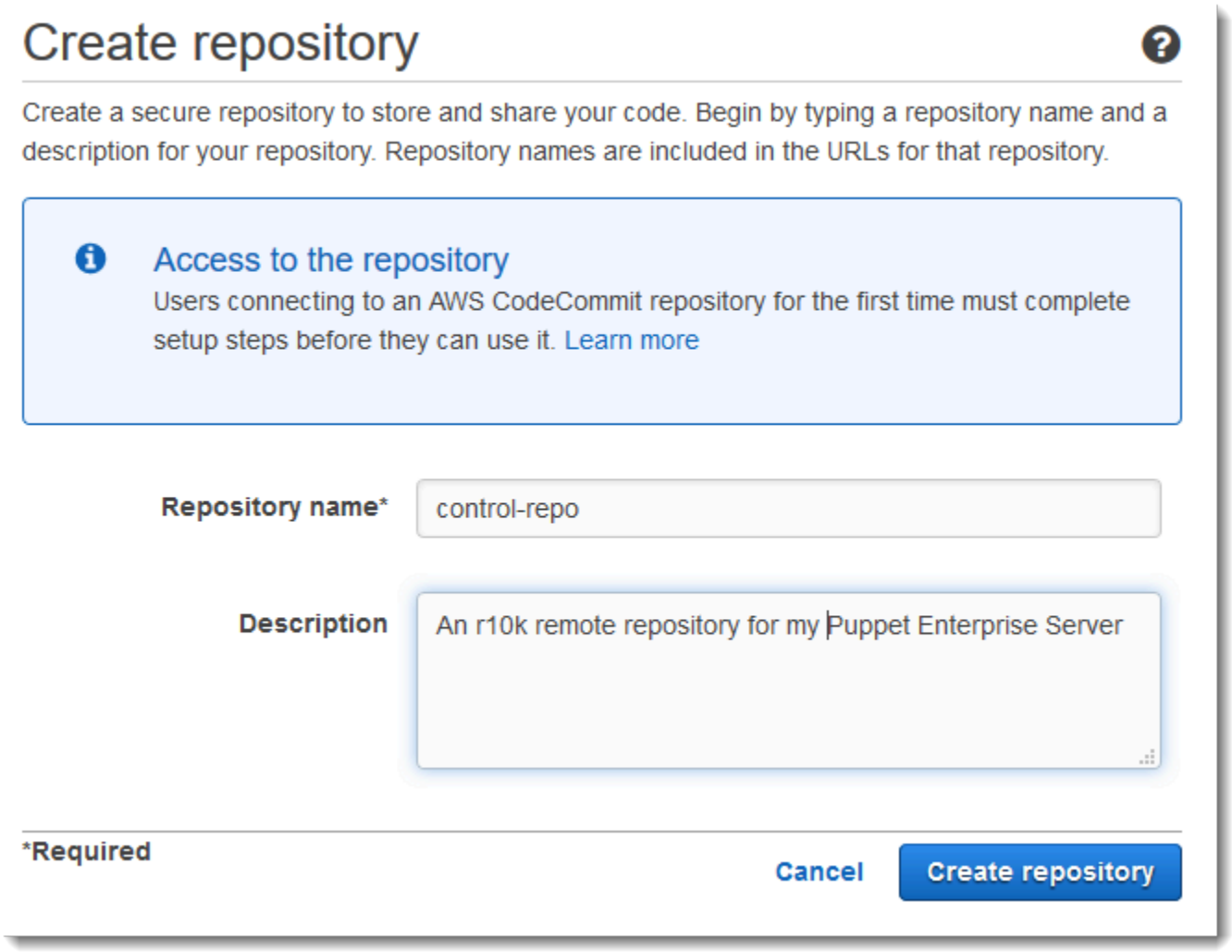
You can create a new repository by using AWS CodeCommit, and use it as your r10k remote control repository. To complete steps in this section, and work with a CodeCommit repository, you need a user that has the permissions provided by the **AWSCodeCommitReadOnly** managed policy.

Topics

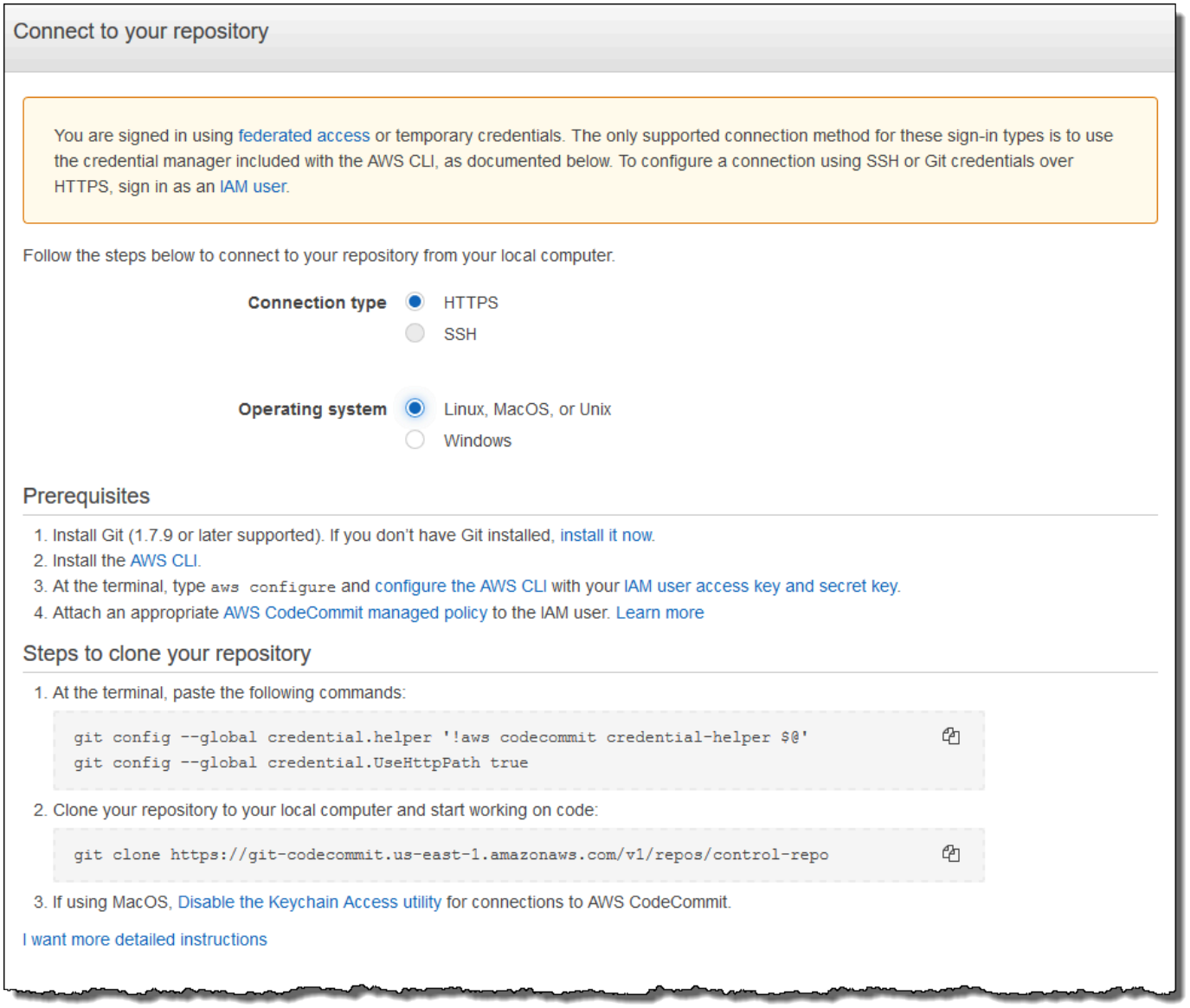
- Step 1: Use CodeCommit as a Repository with an HTTPS connection type
- Step 2: (Optional) Use CodeCommit as a Repository with an SSH connection type

Step 1: Use CodeCommit as a Repository with an HTTPS connection type

1. In the CodeCommit console, create a new repository.



2. Choose **Skip** to skip setting up an Amazon SNS topic.
3. On the **Code** page, choose **Connect to your repository**.
4. On the **Connect to your repository** page, choose **HTTPS** as the **Connection type**, and choose your operating system.



In the **Steps to clone your repository** area, your `git clone` URL should resemble the following: `https://git-codecommit.`*`region`*`.amazonaws.com/v1/repos/control-repo`. Copy this URL to a convenient place for use in Puppet server setup.

5. Close the **Connect to your repository** page, and return to the OpsWorks for Puppet Enterprise server setup.

6. Paste the URL that you copied in Step 4 in the **r10k remote** string box in the **Configure credentials** page of the Puppet master setup wizard. Leave the **r10k private key** box empty. Finish creating and launching your Puppet master.

7. In the IAM console, attach the **AWSCodeCommitReadOnly** policy to the instance profile role of your Puppet master. For more information about how to add a permissions policy to an IAM role, see Adding IAM identity permissions (console) in the *IAM User Guide*.
8. Follow the steps in Setup for HTTPS users using Git credentials in the *AWS CodeCommit User Guide* to push your existing `control-repo` content to the new CodeCommit repository.
9. Now, you can continue by following the instructions in the section called "Finish Configuration", and use the Starter Kit to deploy code to your Puppet master. The following command is an example.

```
puppet-code deploy --all --wait --config-file .config/puppet-code.conf
```

Step 2: (Optional) Use CodeCommit as a Repository with an SSH connection type

You can configure an AWS CodeCommit r10k remote control repository to use SSH key pair authentication. The following prerequisites must be completed before you start this procedure.

- You must have launched your OpsWorks for Puppet Enterprise server with an HTTPS control repository as described in the preceding section, the section called "Step 1: Use CodeCommit as a Repository with an HTTPS connection type". This must be completed first so you can upload the required configuration to the Puppet master.
- Be sure you have an user with the **AWSCodeCommitReadOnly** managed policy attached. For more information about how to create a user, see Creating an IAM user in your AWS account in the *IAM User Guide*.
- Create and associate an SSH key with your user. Follow instructions for creating a public/private key pair with `ssh-keygen` in Step 3: Configure credentials on Linux, macOS, or Unix in the *AWS CodeCommit User Guide*.

1. In an AWS CLI session, run the following command to upload the private key file contents to AWS Systems Manager Parameter Store. Your OpsWorks for Puppet Enterprise server queries this parameter to get a required certificate file. Replace *`private_key_file`* with the path to your SSH private key file.

```
aws ssm put-parameter --name puppet_user_pk --type String --value
 "`cat private_key_file`"
```

2. Add Systems Manager Parameter Store permissions to your Puppet master.

a. Open the IAM console at https://console.aws.amazon.com/iam/.

b. In the left navigation pane, choose **Roles**.

c. Choose **aws-opsworks-cm-ec2-role**.

d. On the **Permissions** tab, choose **Attach policies**.

e. In the **Search** bar, enter **`AmazonSSMManagedInstanceCore`**.

f. In the search results, choose **AmazonSSMManagedInstanceCore**.

g. Choose **Attach policy**.

3. Create the configuration file manifest. If you are using the `control-repo-example` repository provided in the starter kit, create the following files in the locations shown in the example repository. Otherwise, create them according to your own control repository structure. Replace the *`IAM_USER_SSH_KEY`* value with the SSH key ID you created in the prerequisites for this procedure.

```
control-repo-example/site/profile/manifests/codecommit.pp
```

```
class profile::codecommit {
  $configfile = @(CONFIGFILE)
      Host git-codecommit.*.amazonaws.com
      User IAM_USER_SSH_KEY
      IdentityFile /etc/puppetlabs/puppetserver/ssh/codecommit.rsa
      StrictHostKeyChecking=no
      | CONFIGFILE

  # Replace REGION with the correct region for your server.
  $command = @(COMMAND)
      aws ssm get-parameters \
      --region REGION \
      --names puppet_user_pk \
      --query "Parameters[0].Value" \
      --output text >| /etc/puppetlabs/puppetserver/ssh/codecommit.rsa
      | COMMAND

  $dirs = [
            '/opt/puppetlabs/server/data/puppetserver/.ssh',
            '/etc/puppetlabs/puppetserver/ssh',
          ]

  file { $dirs:
```

```
    ensure => 'directory',
    group  => 'pe-puppet',
    owner  => 'pe-puppet',
    mode   => '0750',
  }

  file { 'ssh-config':
    path    => '/opt/puppetlabs/server/data/puppetserver/.ssh/config',
    require => File[$dirs],
    content => $configfile,
    group   => 'pe-puppet',
    owner   => 'pe-puppet',
    mode    => '0600',
  }

  exec { 'download-codecommit-certificate':
    command => $command,
    require => File[$dirs],
    creates => '/etc/puppetlabs/puppetserver/ssh/codecommit.rsa',
    path    => '/bin',
    cwd     => '/etc/puppetlabs',
  }

  file { 'private-key-permissions':
    subscribe => Exec['download-codecommit-certificate'],
    path      => '/etc/puppetlabs/puppetserver/ssh/codecommit.rsa',
    group     => 'pe-puppet',
    owner     => 'pe-puppet',
    mode      => '0600',
  }
}
```

4. Push your control repository to CodeCommit. Run the following commands to push the new manifest file to your repository.

```
git add ./site/profile/manifests/codecommit.pp
git commit -m 'Configuring for SSH connection to CodeCommit'
git push origin production
```

5. Deploy the manifest files. Run the following commands to deploy the updated configuration to your OpsWorks for Puppet Enterprise server. Replace *STARTER_KIT_DIRECTORY* with the path to your Puppet configuration files.

```
cd STARTER_KIT_DIRECTORY

puppet-access login --config-file .config/puppetlabs/client-tools/puppet-
access.conf

puppet-code deploy --all --wait \
--config-file .config/puppet-code.conf \
--token-file .config/puppetlabs/token
```

6. Update the OpsWorks for Puppet Enterprise server's classification. By default, the Puppet agent runs on nodes (including the master) every 30 minutes. To avoid waiting, you can manually run the agent on the Puppet master. Running the agent picks up the new manifest file.

   a. Sign in to the Puppet Enterprise console.

   b. Choose **Classification**.

   c. Expand **PE Infrastructure**.

   d. Choose **PE Master**.

   e. On the **Configuration** tab, enter **`profile::codecommit`** in **Add new class**.

      The new class, `profile::codecommit`, might not appear immediately after running `puppet-code deploy`. Choose **Refresh** on this page if it does not appear.

   f. Choose **Add class**, and then choose **Commit 1 change**.

   g. Manually run the Puppet agent on the OpsWorks for Puppet Enterprise server. Choose **Nodes**, choose your server in the list, choose **Run Puppet**, and then choose **Run**.

7. In the Puppet Enterprise console, change the repository URL to use SSH instead of HTTPS. The configuration you perform in these steps is saved during the OpsWorks for Puppet Enterprise backup and restoration process, so you do not need to manually change the repository configuration after maintenance activities.

   a. Choose **Classification**.

   b. Expand **PE Infrastructure**.

   c. Choose **PE Master**.

   d. On the **Configuration** tab, find the `puppet_enterprise::profile::master` class.

   e. Choose **Edit** next to the `r10k_remote` parameter.

f. Replace the HTTPS URL with the SSH URL for your repository, and then choose **Commit 1 change**.

g. Manually run the Puppet agent on the OpsWorks for Puppet Enterprise server. Choose **Nodes**, choose your server in the list, choose **Run Puppet**, and then choose **Run**.

Create an AWS OpsWorks for Puppet Enterprise Master by using AWS CloudFormation

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

AWS OpsWorks for Puppet Enterprise lets you run a Puppet Enterprise server in AWS. You can provision a Puppet Enterprise master server in about 15 minutes.

Starting May 3, 2021, OpsWorks for Puppet Enterprise stores some Puppet Enterprise server attributes in AWS Secrets Manager. For more information, see Integration with AWS Secrets Manager.

The following walkthrough helps you create a Puppet master in OpsWorks for Puppet Enterprise by creating a stack in AWS CloudFormation.

Topics

- Prerequisites
- Create a Puppet Enterprise Master in AWS CloudFormation

Prerequisites

Before you create a new Puppet master, create the resources outside of OpsWorks for Puppet Enterprise that you'll need to access and manage your Puppet master. For more information, see Prerequisites in the Getting Started section of this guide.

If you are creating a server that uses a custom domain, you need a custom domain, certificate, and private key. You must specify values for all three of these parameters in your AWS CloudFormation template. For more information about requirements for the `CustomDomain`, `CustomCertificate`, and `CustomPrivateKey` parameters, see CreateServer in the *OpsWorks CM API Reference*.

Review the OpsWorks-CM section of the *AWS CloudFormation User Guide* Template Reference to learn about the supported and required values in the AWS CloudFormation template that you use to create your server.

Create a Puppet Enterprise Master in AWS CloudFormation

This section describes how to use an AWS CloudFormation template to build a stack that creates an OpsWorks for Puppet Enterprise master server. You can do this by using the AWS CloudFormation console or the AWS CLI. An example AWS CloudFormation template is available for you to use to build an OpsWorks for Puppet Enterprise server stack. Be sure to update the example template with your own server name, IAM roles, instance profile, server description, backup retention count, maintenance options, and optional tags. If your server will use a custom domain, you must specify values for the `CustomDomain`, `CustomCertificate`, and `CustomPrivateKey` parameters in your AWS CloudFormation template. For more information about these options, see the section called "Create a Puppet Enterprise Master by using the AWS Management Console" in the Getting Started section of this guide.

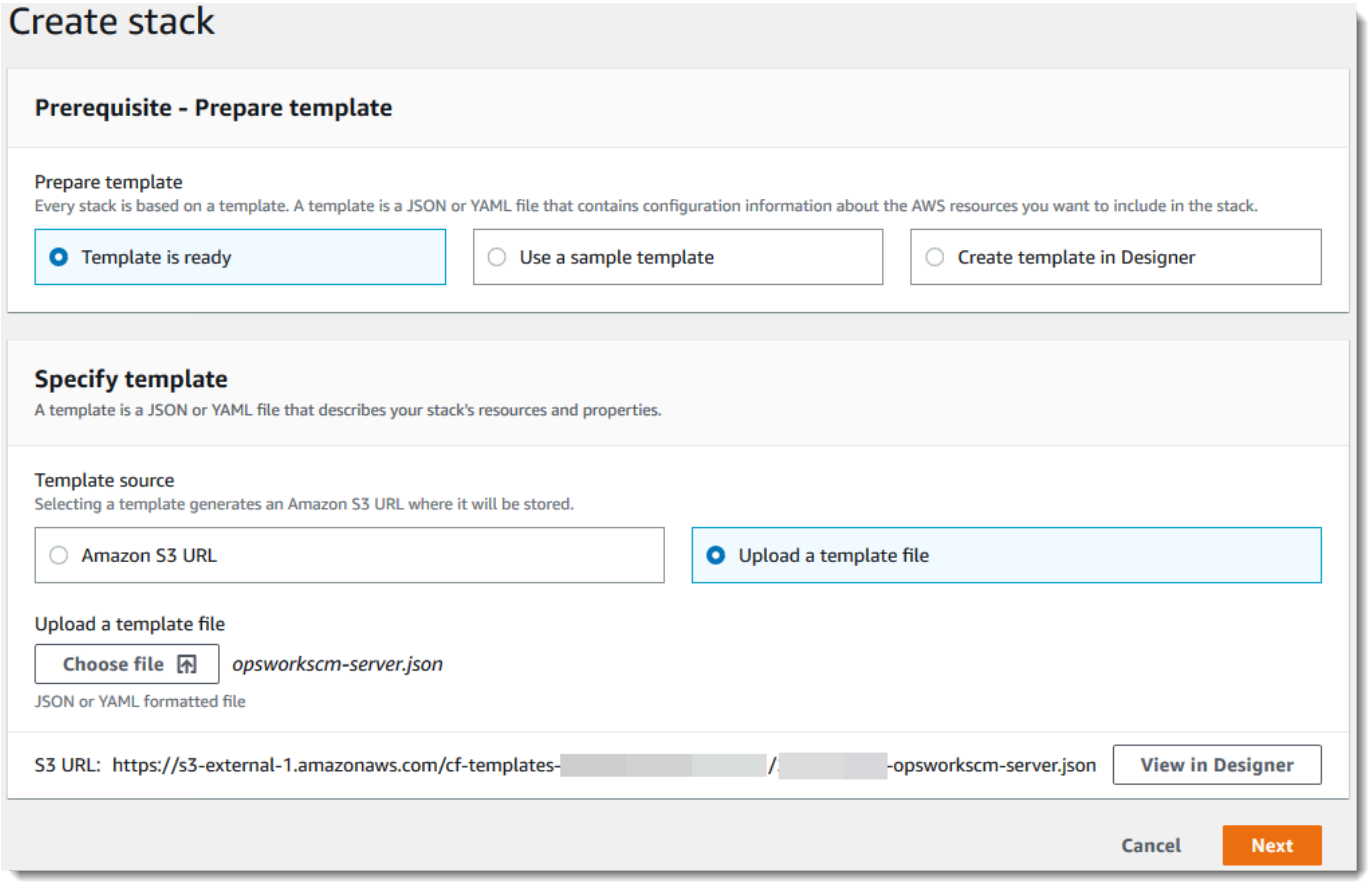
Topics

- Create a Puppet Enterprise Master by using AWS CloudFormation (Console)
- Create a Puppet Enterprise Master by using AWS CloudFormation (CLI)

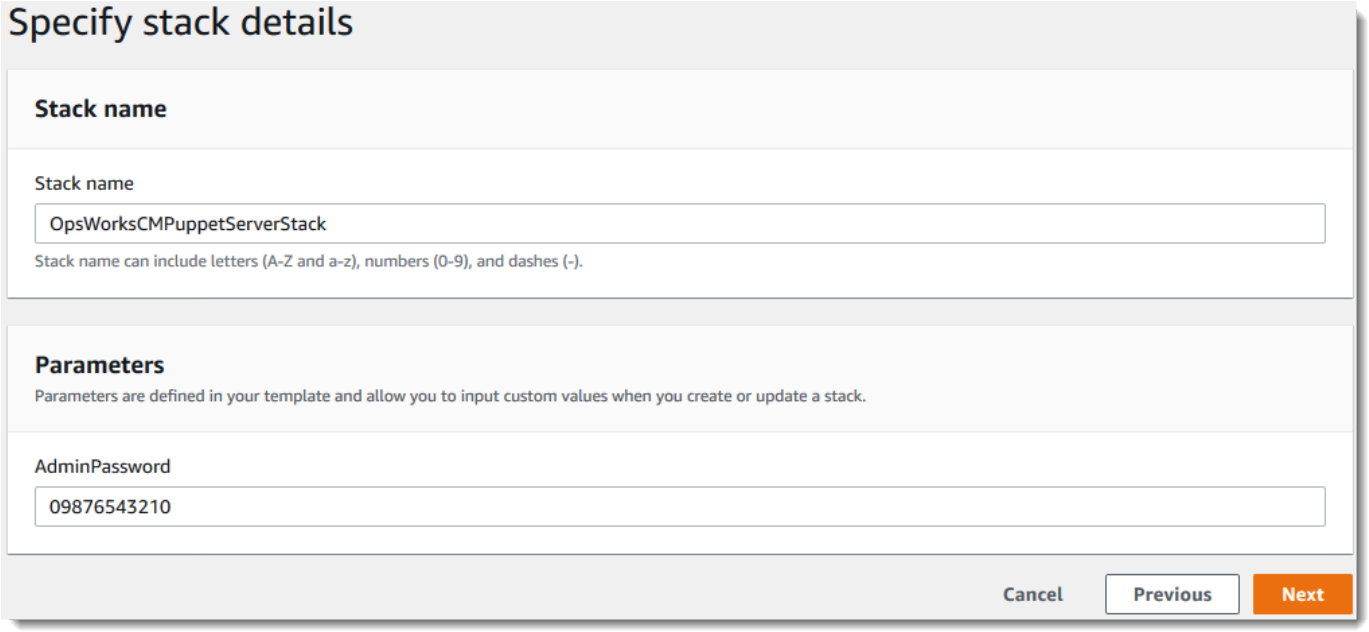
Create a Puppet Enterprise Master by using AWS CloudFormation (Console)

1. Sign in to the AWS Management Console and open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. On the AWS CloudFormation home page, choose **Create stack**.
3. In **Prerequisite - Prepare template**, if you are using the example AWS CloudFormation template, choose **Template is ready**.
4. In **Specify template**, choose the source of your template. For this walkthrough, choose **Upload a template file**, and upload an AWS CloudFormation template that creates a Puppet Enterprise server. Browse for your template file, and then choose **Next**.

An AWS CloudFormation template can be in either YAML or JSON format. An example AWS CloudFormation template is available for you to use; be sure to replace example values with your own. You can use the AWS CloudFormation template designer to build a new template or validate an existing one. For more information about how to do this, see AWS CloudFormation Designer Interface Overview in the *AWS CloudFormation User Guide*.



5. On the **Specify stack details** page, enter a name for your stack. This won't be the same as the name of your server, it is only a stack name. In the **Parameters** area, enter an administrator password for signing in to the Puppet Enterprise console webpage. The password must use between 8 and 32 ASCII characters. Choose **Next**.



6. On the **Options** page, you can add tags to the server you're creating with the stack, and choose an IAM role for creating resources if you have not already specified an IAM role to use in your template. When you're finished specifying options, choose **Next**. For more information about advanced options such as rollback triggers, see Setting AWS CloudFormation Stack Options in the *AWS CloudFormation User Guide*.
7. On the **Review** page, review your choices. When you are ready to create the server stack, choose **Create**.

   While you are waiting for AWS CloudFormation to create the stack, view the stack creation status. If stack creation fails, review the error messages shown in the console to help you resolve the issues. For more information about troubleshooting errors in AWS CloudFormation stacks, see Troubleshooting Errors in the *AWS CloudFormation User Guide*.

   When server creation is finished, your Puppet master is available on the OpsWorks for Puppet Enterprise home page, with a status of **online**. After the server is online, the Puppet Enterprise console is available on the server's domain, at a URL in the following format: `https://your_server_name-randomID.region.opsworks-cm.io`.

   > **Note**
   >
   > If you specified a custom domain, certificate, and private key for your server, create a CNAME entry in your enterprise's DNS management tool that maps your custom domain to the endpoint that OpsWorks for Puppet Enterprise automatically generated

for the server. You cannot manage the server or connect to the Puppet Enterprise management website for the server until you map the generated endpoint to your custom domain value.
To get the generated endpoint value, run the following AWS CLI command after your server is online:

```
aws opsworks describe-servers --server-name server_name
```

Create a Puppet Enterprise Master by using AWS CloudFormation (CLI)

If your local computer is not already running the AWS CLI, download and install the AWS CLI by following installation instructions in the *AWS Command Line Interface User Guide*. This section does not describe all parameters that you can use with the **create-stack** command. For more information about **create-stack** parameters, see **create-stack** in the *AWS CLI Reference*.

1. Be sure to complete the Prerequisites for creating an OpsWorks for Puppet Enterprise master.
2. Create a service role and an instance profile.

   Find and copy the ARNs of service roles in your account.

   ```
   aws iam list-roles --path-prefix "/service-role/" --no-paginate
   ```

   In the results of the `list-roles` command, look for service role and instance profile entries that resemble the following. Make a note of the ARNs of the service role and instance profile, and add them to the AWS CloudFormation template that you are using to create your Puppet master server stack.
3. Create the OpsWorks for Puppet Enterprise master by running the **create-stack** command again.

   - Replace *`stack_name`* with the name of your stack. This is the name of the AWS CloudFormation stack, not your Puppet master. The Puppet master name is the value of `ServerName` in your AWS CloudFormation template.
   - Replace *`template`* with the path to your template file, and the extension *`yaml or json`* with `.yaml` or `.json` as appropriate.
   - The values for `--parameters` correspond to EngineAttributes from the CreateServer API. For Puppet, the following are user-provided engine attributes to create a server.

r10k engine attributes connect your Puppet master to a code repository to manage the server's environment configuration. For more information about r10k engine attributes, see Managing code with r10k in the Puppet Enterprise documentation.

- PUPPET_ADMIN_PASSWORD, an administrator password for signing in to the Puppet Enterprise console webpage. The password must use between 8 and 32 ASCII characters, and requires at least one upper case letter, one lower case letter, one number, and one special character.
- PUPPET_R10K_REMOTE, the URL of your control repository (for example, ssh://git@your.git-repo.com:user/control-repo.git). Specifying an r10k remote opens TCP port 8170.
- PUPPET_R10K_PRIVATE_KEY. If you are using a private Git repository, add PUPPET_R10K_PRIVATE_KEY to specify an SSH URL and a PEM-encoded private SSH key.

```
aws cloudformation create-stack --stack-name stack_name
 --template-body file://template.yaml or json --parameters
 ParameterKey=AdminPassword,ParameterValue="password"
```

The following is an example.

```
aws cloudformation create-stack --stack-name "OpsWorksCMPuppetServerStack"
 --template-body file://opsworkscm-puppet-server.json --parameters
 ParameterKey=AdminPassword,ParameterValue="09876543210Ab#"
```

The following example specifies r10k engine attributes as parameters, when they are not provided in the AWS CloudFormation template. An example template that includes the r10k engine attributes, `puppet-server-param-attributes.yaml`, is included in the example AWS CloudFormation templates.

```
aws cloudformation create-stack --stack-name MyPuppetStack --
template-body file://puppet-server-param-attributes.yaml --parameters
 ParameterKey=AdminPassword,ParameterValue="superSecret1%3"
 ParameterKey=R10KRemote,ParameterValue="https://www.yourRemote.com"
 ParameterKey=R10KKey,ParameterValue="$(cat puppet-r10k.pem)"
```

The following example specifies r10k engine attributes and their values in the AWS CloudFormation template; the command only needs to point to the template file. The

template specified as the value of `--template-body`, `puppet-server-in-file-attributes.yaml`, is included in the example AWS CloudFormation templates.

```
aws cloudformation create-stack --stack-name MyPuppetStack --template-body file://
puppet-server-in-file-attributes.yaml
```

4. (Optional) To get stack creation status, run the following command.

```
aws cloudformation describe-stacks --stack-name stack_name
```

5. When stack creation has finished, go on to the next section, the section called "Finish Configuration". If stack creation fails, review the error messages shown in the console to help you resolve the issues. For more information about troubleshooting errors in AWS CloudFormation stacks, see Troubleshooting Errors in the *AWS CloudFormation User Guide*.

Update an OpsWorks for Puppet Enterprise Server to Use a Custom Domain

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section describes how to update an existing OpsWorks for Puppet Enterprise server to use a custom domain and certificate by using a backup of the server to create a new server. Essentially, you are copying an existing OpsWorks for Puppet Enterprise 2.0 server by creating a new server from a backup, then configuring the new server to use a custom domain, certificate, and private key.

Topics

- Prerequisites
- Limitations
- Update a Server to Use a Custom Domain

- See Also

Prerequisites

The following are requirements for updating an existing OpsWorks for Puppet Enterprise server to use a custom domain and certificate.

- The server that you want to update (or copy) must be running Puppet Enterprise 2019.8.5.
- Decide which backup you want to use to create a new server. You must have at least one backup available of the server that you want to update. For more information about backups in OpsWorks for Puppet Enterprise, see Back Up an OpsWorks for Puppet Enterprise Server.
- Have ready the service role and instance profile ARNs that you used to create the existing server that is the source of your backup.
- Be sure that you are running the most current release of the AWS CLI. For more information about updating your AWS CLI tools, see Installing the AWS CLI in the *AWS Command Line Interface User Guide*.

Limitations

When you update an existing server by creating a new server from a backup, the new server cannot be exactly the same as the existing OpsWorks for Puppet Enterprise server.

- You can only complete this procedure by using the AWS CLI or one of the AWS SDKs. You cannot create a new server from a backup by using the AWS Management Console.
- The new server cannot use the same name as the existing server within an account, and within an AWS Region. The name must be different from the existing server that you used as the source of the backup.
- Nodes that were attached to the existing server are not managed by the new server. You must do one of the following.
  - Attach different nodes, because nodes cannot be managed by more than one Puppet master.
  - Migrate the nodes from the existing server (the source of the backup) to the new server and the new custom domain endpoint. For more information about how to migrate nodes, see the Puppet Enterprise documentation.

Update a Server to Use a Custom Domain

To update an existing Puppet master, you make a copy of it by running the `create-server` command, adding parameters to specify a backup, a custom domain, a custom certificate, and a custom private key.

1. If you do not have service role or instance profile ARNs available to specify in your `create-server` command, follow steps 1-5 in Create a Chef Automate server by using the AWS CLI to create a service role and instance profile that you can use.
2. If you have not already done so, find the backup of the existing Puppet master on which you want to base a new server with a custom domain. Run the following command to show information about all OpsWorks for Puppet Enterprise backups in your account, and in a region. Be sure to note the ID of the backup that you want to use.

   ```
   aws opsworks-cm --region region name describe-backups
   ```

3. Create the OpsWorks for Puppet Enterprise server by running the `create-server` command.

   - The `--engine` value is `Puppet`, `--engine-model` is `Monolithic`, and `--engine-version` is `2019` or `2017`.
   - The server name must be unique within your AWS account, within each region. Server names must start with a letter; then letters, numbers, or hyphens (-) are allowed, up to a maximum of 40 characters.
   - Use the instance profile ARN and service role ARN that you copied in Steps 3 and 4.
   - Valid instance types are `c4.large`, `c4.xlarge`, or `c4.2xlarge`. For more information about the specifications of these instance types, see Instance Types in the *Amazon EC2 User Guide*.
   - The `--engine-attributes` parameter is optional; if you don't specify a Puppet administrator password, the server creation process generates one for you. If you add `--engine-attributes`, specify a `PUPPET_ADMIN_PASSWORD`, an administrator password for signing in to the Puppet Enterprise console webpage. The password must use between 8 and 32 ASCII characters.
   - An SSH key pair is optional, but can help you connect to your Puppet master if you need to reset the console administrator password. For more information about creating an SSH key pair, see Amazon EC2 Key Pairs in the *Amazon EC2 User Guide*.

- To use a custom domain, add the following parameters to your command. Otherwise, the Puppet master creation process automatically generates an endpoint for you. All three parameters are required to configure a custom domain. For information about additional requirements for using these parameters, see CreateServer in the OpsWorks CM API Reference.
  - `--custom-domain` - An optional public endpoint of a server, such as `https://aws.my-company.com`.
  - `--custom-certificate` - A PEM-formatted HTTPS certificate. The value can be be a single, self-signed certificate, or a certificate chain.
  - `--custom-private-key` - A private key in PEM format for connecting to the server by using HTTPS. The private key must not be encrypted; it cannot be protected by a password or passphrase.
- Weekly system maintenance is required. Valid values must be specified in the following format: `DDD:HH:MM`. The specified time is in coordinated universal time (UTC). If you do not specify a value for `--preferred-maintenance-window`, the default value is a random, one-hour period on Tuesday, Wednesday, or Friday.
- Valid values for `--preferred-backup-window` must be specified in one of the following formats: `HH:MM` for daily backups, or `DDD:HH:MM` for weekly backups. The specified time is in UTC. The default value is a random, daily start time. To opt out of automatic backups, add the parameter `--disable-automated-backup` instead.
- For `--security-group-ids`, enter one or more security group IDs, separated by a space.
- For `--subnet-ids`, enter a subnet ID.

```
aws opsworks-cm create-server --engine "Puppet" --engine-model "Monolithic"
 --engine-version "2019" --server-name "server_name" --instance-profile-arn
 "instance_profile_ARN" --instance-type "instance_type" --engine-attributes
 '{"PUPPET_ADMIN_PASSWORD":"ASCII_password"}' --key-pair "key_pair_name" --
preferred-maintenance-window "ddd:hh:mm" --preferred-backup-window "ddd:hh:mm"
 --security-group-ids security_group_id1 security_group_id2 --service-role-arn
 "service_role_ARN" --subnet-ids subnet_ID
```

The following example creates a Puppet master that uses a custom domain.

```
aws opsworks-cm create-server \
    --engine "Puppet" \
```

```
    --engine-model "Monolithic" \
    --engine-version "2019" \
    --server-name "puppet-02" \
    --instance-profile-arn "arn:aws:iam::1019881987024:instance-profile/aws-
opsworks-cm-ec2-role" \
    --instance-type "c4.large" \
    --engine-attributes '{"PUPPET_ADMIN_PASSWORD":"zZZzDj2DLYXSZFRv1d"}' \
    --custom-domain "my-puppet-master.my-corp.com" \
    --custom-certificate "-----BEGIN CERTIFICATE----- EXAMPLEqEXAMPLE== -----END
 CERTIFICATE-----" \
    --custom-private-key "-----BEGIN RSA PRIVATE KEY----- EXAMPLEqEXAMPLE= -----END
 RSA PRIVATE KEY-----" \
    --key-pair "amazon-test"
    --preferred-maintenance-window "Mon:08:00" \
    --preferred-backup-window "Sun:02:00" \
    --security-group-ids sg-b00000001 sg-b0000008 \
    --service-role-arn "arn:aws:iam::044726508045:role/service-role/aws-opsworks-
cm-service-role" \
    --subnet-ids subnet-383daa71
```

4. OpsWorks for Puppet Enterprise takes about 15 minutes to create a new server. In the output of the `create-server` command, copy the value of the `Endpoint` attribute. The following is an example.

   ```
   "Endpoint": "puppet-2019-exampleexample.opsworks-cm.us-east-1.amazonaws.com"
   ```

   Do not dismiss the output of the `create-server` command or close your shell session, because the output can contain important information that is not shown again. To get passwords and the starter kit from the `create-server` results, go on to the next step.

5. If you opted to have OpsWorks for Puppet Enterprise generate a password for you, you can extract it in a usable format from the `create-server` results by using a JSON processor such as jq. After you install jq, you can run the following commands to extract the Puppet administrator password and starter kit. If you did not provide your own password in Step 3, be sure to save the extracted administrator password in a convenient but secure location.

   ```
   #Get the Puppet password:
   cat resp.json | jq -r '.Server.EngineAttributes[] | select(.Name ==
    "PUPPET_ADMIN_PASSWORD") | .Value'

   #Get the Puppet Starter Kit:
   ```

```
cat resp.json | jq -r '.Server.EngineAttributes[] | select(.Name ==
 "PUPPET_STARTER_KIT") | .Value' | base64 -D > starterkit.zip
```

> **Note**
>
> You cannot regenerate a new Puppet master starter kit in the AWS Management Console. When you create a Puppet master by using the AWS CLI, run the preceding `jq` command to save the base64-encoded starter kit in the `create-server` results as a ZIP file.

6. Optionally, if you did not extract the starter kit from `create-server` command results, you can download a new starter kit from the server's Properties page in the OpsWorks for Puppet Enterprise console.
7. If you are not using a custom domain, go on to the next step. If you are using a custom domain with the server, create a CNAME entry in your enterprise's DNS management tool to point your custom domain to the OpsWorks for Puppet Enterprise endpoint that you copied in step 4. You cannot reach or sign in to a server with a custom domain until you complete this step.
8. When the server creation process is finished, go on to Configure the Puppet Master Using the Starter Kit.

See Also

- Create a Puppet Enterprise Master by using the AWS CLI
- Back Up and Restore an OpsWorks for Puppet Enterprise Server
- CreateServer in the OpsWorks CM API Reference
- **create-server** in the *AWS CLI Command Reference*

Working with Tags on AWS OpsWorks for Puppet Enterprise Resources

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have

questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Tags are words or phrases that act as metadata for identifying and organizing your AWS resources. In OpsWorks for Puppet Enterprise, a resource can have up to 50 user-applied tags. Each tag consists of a key and one optional value. You can apply tags to the following resources in OpsWorks for Puppet Enterprise:

- OpsWorks for Puppet Enterprise servers
- Backups of OpsWorks for Puppet Enterprise servers

Tags on AWS resources can help you track costs, control access to resources, group resources for automating tasks, or organize resources by purpose or lifecycle stage. For more information about the benefits of tags, see AWS Tagging Strategies in AWS Answers and Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

To use tags to control access to OpsWorks for Puppet Enterprise servers or backups, create or edit policy statements in AWS Identity and Access Management (IAM). For more information, see Controlling Access to AWS Resources Using Resource Tags in the *AWS Identity and Access Management User Guide*.

When you apply tags to an OpsWorks for Puppet Enterprise master, the tags are also applied to the master's backups, the Amazon S3 bucket that stores the backups, the master's Amazon EC2 instance, secrets for the master that are stored in AWS Secrets Manager, and the Elastic IP address used by the master. Tags are not propagated to the AWS CloudFormation stack that OpsWorks uses to create your Puppet master.

Topics

- How Tags Work in AWS OpsWorks for Puppet Enterprise
- Add and Manage Tags in OpsWorks for Puppet Enterprise (Console)
- Add and Manage Tags in OpsWorks for Puppet Enterprise (CLI)
- See Also

How Tags Work in AWS OpsWorks for Puppet Enterprise

In this release, you can add and manage tags by using the OpsWorks CM API or the AWS Management Console. OpsWorks CM also attempts to add tags that you add to a server to the AWS resources that are associated with the server, including the EC2 instance, secrets in Secrets Manager, Elastic IP address, security group, S3 bucket, and backups.

The following table provides an overview of how you add and manage tags in OpsWorks for Puppet Enterprise.

Action	What to use
Add tags to a new OpsWorks for Puppet Enterprise server or a backup that you are creating manually.	• Choose **Create Puppet Enterprise server** and add tags on the **Configure advanced settings** page. • Choose **Create backup** on the **Backups** page for an existing server, and add tags on the **Create a backup of your Puppet Enterprise server** page. • Add a `Tags` parameter to the `CreateServer` or `CreateBackup` commands.
View tags on a resource.	• On the details page for your server, choose **Tags** in the navigation pane. • On the **Backups** page for your server, select a backup, and then choose **Edit backup**. • Run the `ListTagsForResource` command.
Add tags to an existing OpsWorks for Puppet Enterprise server or a backup, regardless of whether the backup was created manually or automatically.	• On the details page for your server, choose **Tags** in the navigation pane, and then choose **Edit**. • On the **Backups** page for your server, select a backup, and then choose **Edit backup**. • Run the `TagResource` command.

Action	What to use
Delete tags from a resource.	• On the details page for your server, choose **Tags** in the navigation pane, and then choose **Edit**. Choose **X** next to tags that you want to delete. • On the **Backups** page for your server, select a backup, and then choose **Edit backup**. Choose **X** next to tags that you want to delete. • Run the `UntagResource` command.

`DescribeServers` and `DescribeBackups` responses do not include tag information. To show tags, use the `ListTagsForResource` API.

Add and Manage Tags in OpsWorks for Puppet Enterprise (Console)

Procedures in this section are performed in the AWS Management Console.

If you add tags, a tag key cannot be empty. The key can be a maximum of 127 characters, and can contain only Unicode letters, numbers, or separators, or the following special characters: `+ - = . _ : / @` A tag value is optional. You can add a tag that has a key, but no value. The value can be a maximum of 255 characters, and can contain only Unicode letters, numbers, or separators, or the following special characters: `+ - = . _ : / @`

Topics

- Add Tags to a New OpsWorks for Puppet Enterprise Server (Console)
- Add Tags to a New Backup (Console)
- Add or View Tags on an Existing Server (Console)
- Add or View Tags on an Existing Backup (Console)
- Delete Tags from a Server (Console)
- Delete Tags from a Backup (Console)

Add Tags to a New OpsWorks for Puppet Enterprise Server (Console)

1. Be sure to complete any prerequisites for creating an OpsWorks for Puppet Enterprise master.
2. Follow steps 1-8 in Create a Puppet Enterprise Master by using the AWS Management Console.
3. After you specify automated backup settings, add tags in the **Tags** area of the **Configure advanced settings** page. You can add a maximum of 50 tags. When you are finished adding tags, choose **Next**.
4. Go on to step 11 of Create a Puppet Enterprise Master by using the AWS Management Console, and review settings you have chosen for the new server.

Add Tags to a New Backup (Console)

1. On the OpsWorks for Puppet Enterprise home page, choose an existing Puppet master.
2. From the server's details page, choose **Backups** in the navigation pane.
3. On the **Backups** page, choose **Create backup**.
4. Add tags. Choose **Create** when you are finished adding tags.

Add or View Tags on an Existing Server (Console)

1. On the OpsWorks for Puppet Enterprise home page, choose an existing Puppet master to open its details page.
2. Choose **Tags** in the navigation pane, or at the bottom of the details page, choose **View all tags**.
3. On the **Tags** page, choose **Edit**.
4. Add or edit tags on the server. Choose **Save** when you are finished.

> **Note**
>
> Be aware that changing tags on your Puppet master also changes tags on resources that are associated with the server, such as the EC2 instance, Elastic IP address, security group, S3 bucket, and backups.

Add or View Tags on an Existing Backup (Console)

1. On the OpsWorks for Puppet Enterprise home page, choose an existing Puppet master to open its details page.
2. Choose **Backups** in the navigation pane, or in the **Recent backups** area of the details page, choose **View all backups**.
3. On the **Backups** page, choose a backup to manage, and then choose **Edit backup**.
4. Add or edit tags on the backup. Choose **Update** when you are finished.

Delete Tags from a Server (Console)

1. On the OpsWorks for Puppet Enterprise home page, choose an existing Puppet master to open its details page.
2. Choose **Tags** in the navigation pane, or at the bottom of the details page, choose **View all tags**.
3. On the **Tags** page, choose **Edit**.
4. Choose **X** next to a tag to delete the tag. Choose **Save** when you are finished.

> **Note**
>
> Be aware that changing tags on your Puppet master also changes tags on resources that are associated with the server, such as the EC2 instance, Elastic IP address, security group, S3 bucket, and backups.

Delete Tags from a Backup (Console)

1. On the OpsWorks for Puppet Enterprise home page, choose an existing Puppet master to open its details page.
2. Choose **Backups** in the navigation pane, or in the **Recent backups** area of the details page, choose **View all backups**.
3. On the **Backups** page, choose a backup to manage, and then choose **Edit backup**.
4. Choose **X** next to a tag to delete the tag. Choose **Update** when you are finished.

Add and Manage Tags in OpsWorks for Puppet Enterprise (CLI)

Procedures in this section are performed in the AWS CLI. Be sure that you are running the latest release of the AWS CLI before you start working with tags. For more information about installing or updating the AWS CLI, see Installing the AWS CLI in the *AWS Command Line Interface User Guide*.

If you add tags, a tag key cannot be empty. The key can be a maximum of 127 characters, and can contain only Unicode letters, numbers, or separators, or the following special characters: `+ - = . _ : / @` A tag value is optional. You can add a tag that has a key, but no value. The value can be a maximum of 255 characters, and contain only Unicode letters, numbers, or separators, or the following special characters: `+ - = . _ : / @`

Topics

- Add Tags to a New OpsWorks for Puppet Enterprise Server (CLI)
- Add Tags to a New Backup (CLI)
- Add Tags to Existing Servers or Backups (CLI)
- List Resource Tags (CLI)
- Delete Tags from a Resource (CLI)

Add Tags to a New OpsWorks for Puppet Enterprise Server (CLI)

You can use the AWS CLI to add tags when you create an OpsWorks for Puppet Enterprise server. This procedure does not describe in full how to create a server. For detailed information about how to create an OpsWorks for Puppet Enterprise server by using the AWS CLI, see Create a Puppet Enterprise Master by using the AWS CLI in this guide. You can add up to 50 tags to a server.

1. Be sure to complete any prerequisites for creating an OpsWorks for Puppet Enterprise server.
2. Complete steps 1-4 of Create a Puppet Enterprise Master by using the AWS CLI.
3. For step 5, when you run the `create-server` command, add the `--tags` parameter to the command, as shown in the following example.

   ```
   aws opsworks-cm create-server ... --tags Key=Key1,Value=Value1
    Key=Key2,Value=Value2
   ```

   The following is an example showing only the tags portion of the `create-server` command.

```
aws opsworks-cm create-server ... --tags Key=Stage,Value=Production
 Key=Department,Value=Marketing
```

4. Complete the remaining steps in Create a Puppet Enterprise Master by using the AWS CLI. To verify that your tags were added to the new server, follow steps in List Resource Tags (CLI) in this topic.

Add Tags to a New Backup (CLI)

You can use the AWS CLI to add tags when you create a new, manual backup of an OpsWorks for Puppet Enterprise server. This procedure does not describe in full how to create a manual backup. For detailed information about how to create a manual backup, see "To perform a manual backup in the AWS CLI" in Back Up an OpsWorks for Puppet Enterprise Server. You can add up to 50 tags to a backup. If a server has tags, new backups are automatically tagged with the server's tags.

By default, when you create a new OpsWorks for Puppet Enterprise server, automated backups are enabled. You can add tags to an automated backup by running the `tag-resource` command, described in Add Tags to Existing Servers or Backups (CLI) in this topic.

- To add tags to a manual backup as you're creating the backup, run the following command. Only the tags portion of the command is shown. For an example of the full `create-backup` command, see "To perform a manual backup in the AWS CLI" in Back Up an OpsWorks for Puppet Enterprise Server.

  ```
  aws opsworks-cm create-backup ... --tags Key=Key1,Value=Value1
   Key=Key2,Value=Value2
  ```

  The following example shows only the tags portion of the `create-backup` command.

  ```
  aws opsworks-cm create-backup ... --tags Key=Stage,Value=Production
   Key=Department,Value=Marketing
  ```

Add Tags to Existing Servers or Backups (CLI)

You can run the `tag-resource` command to add tags to existing OpsWorks for Puppet Enterprise servers or backups (whether the backups were created automatically or manually). Specify the Amazon Resource Number (ARN) of a target resource to add tags to it.

1. To get the ARN of the resource to which you want to apply tags:

   - For a server, run `describe-servers --server-name` *`server_name`*. The results of the command show the server ARN.
   - For a backup, run `describe-backups --backup-id` *`backup_ID`*. The results of the command show the backup ARN. You can also run `describe-backups --server-name` *`server_name`* to show information about all backups for a specific OpsWorks for Puppet Enterprise server.

   The following example shows only the `ServerArn` in results of a `describe-servers --server-name opsworks-cm-test` command. The `ServerArn` value is added to a `tag-resource` command to add tags to the server.

   ```
   {
       "Servers": [
           {
               ...
               "ServerArn": "arn:aws:opsworks-cm:us-west-2:123456789012:server/
   opsworks-cm-test/EXAMPLEd-66b0-4196-8274-d1a2bEXAMPLE"
           }
       ]
   }
   ```

2. Run the `tag-resource` command with the ARN that you returned in step 1.

   ```
   aws opsworks-cm tag-resource --resource-arn "server_or_backup_ARN" --tags
    Key=Key1,Value=Value1 Key=Key2,Value=Value2
   ```

   The following is an example.

   ```
   aws opsworks-cm tag-resource --resource-arn "arn:aws:opsworks-cm:us-
   west-2:123456789012:server/opsworks-cm-test/EXAMPLEd-66b0-4196-8274-d1a2bEXAMPLE"
    --tags Key=Stage,Value=Production Key=Department,Value=Marketing
   ```

3. To verify that tags were added successfully, go on to the next procedure, List Resource Tags (CLI).

List Resource Tags (CLI)

You can run the `list-tags-for-resource` command to show the tags that are attached to OpsWorks for Puppet Enterprise servers or backups. Specify the ARN of a target resource to view its tags.

1. To get the ARN of the resource for which you want to list tags:
   - For a server, run `describe-servers --server-name` *`server_name`*. The results of the command show the server ARN.
   - For a backup, run `describe-backups --backup-id` *`backup_ID`*. The results of the command show the backup ARN. You can also run `describe-backups --server-name` *`server_name`* to show information about all backups for a specific OpsWorks for Puppet Enterprise server.
2. Run the `list-tags-for-resource` command with the ARN that you returned in step 1.

   ```
   aws opsworks-cm list-tags-for-resource --resource-arn "server_or_backup_ARN"
   ```

   The following is an example.

   ```
   aws opsworks-cm tag-resource --resource-arn "arn:aws:opsworks-cm:us-
   west-2:123456789012:server/opsworks-cm-test/EXAMPLEd-66b0-4196-8274-d1a2bEXAMPLE"
   ```

   If there are tags on the resource, the command returns results like the following.

   ```
   {
       "Tags": [
           {
               "Key": "Stage",
               "Value": "Production"
           },
           {
               "Key": "Department",
               "Value": "Marketing"
           }
       ]
   }
   ```

Delete Tags from a Resource (CLI)

You can run the `untag-resource` command to delete tags from OpsWorks for Puppet Enterprise servers or backups. If the resource is deleted, tags on the resource are also deleted. Specify the Amazon Resource Number (ARN) of a target resource to remove tags from it.

1. To get the ARN of the resource from which you want to remove tags:
   - For a server, run `describe-servers --server-name` *`server_name`*. The results of the command show the server ARN.
   - For a backup, run `describe-backups --backup-id` *`backup_ID`*. The results of the command show the backup ARN. You can also run `describe-backups --server-name` *`server_name`* to show information about all backups for a specific OpsWorks for Puppet Enterprise server.
2. Run the `untag-resource` command with the ARN that you returned in step 1. Specify only the tags that you want to delete.

   ```
   aws opsworks-cm untag-resource --resource-arn "server_or_backup_ARN" --tags
    Key=Key1,Value=Value1 Key=Key2,Value=Value2
   ```

   In this example, the `untag-resource` command removes only the tag with a key of `Stage` and a value of `Production`.

   ```
   aws opsworks-cm untag-resource --resource-arn "arn:aws:opsworks-cm:us-
   west-2:123456789012:server/opsworks-cm-test/EXAMPLEd-66b0-4196-8274-d1a2bEXAMPLE"
    --tags Key=Stage,Value=Production
   ```

3. To verify that tags were deleted successfully, follow steps in List Resource Tags (CLI) in this topic.

See Also

- Create a Puppet Enterprise Master by using the AWS CLI
- Back Up an OpsWorks for Puppet Enterprise Server
- AWS Tagging Strategies
- Controlling Access to AWS Resources Using Resource Tags in the *AWS Identity and Access Management User Guide*

- Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*
- CreateBackup in the *OpsWorks CM API Reference*
- CreateServer in the *OpsWorks CM API Reference*
- TagResource in the *OpsWorks CM API Reference*
- ListTagsForResource in the *OpsWorks CM API Reference*
- UntagResource in the *OpsWorks CM API Reference*

Back Up and Restore an OpsWorks for Puppet Enterprise Server

Important

The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section describes how to back up and restore an OpsWorks for Puppet Enterprise server.

Topics

- Back Up an OpsWorks for Puppet Enterprise Server
- Restore an OpsWorks for Puppet Enterprise Server from a Backup

Back Up an OpsWorks for Puppet Enterprise Server

Important

The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can define a daily or weekly recurring OpsWorks for Puppet Enterprise server backup, and have the service store the backups in Amazon Simple Storage Service (Amazon S3) on your behalf. Alternatively, you can make manual backups on demand.

Because backups are stored in Amazon S3, they incur additional fees. You can define a backup retention period of up to 30 generations. You can submit a service request to have that limit changed by using AWS support channels. Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

You can add tags to backups of an OpsWorks for Puppet Enterprise master. If you have added tags to an OpsWorks for Puppet Enterprise master, automated backups of the Puppet master inherit those tags. For more information about how to add and manage tags on backups, see Working with Tags on AWS OpsWorks for Puppet Enterprise Resources in this guide.

Topics

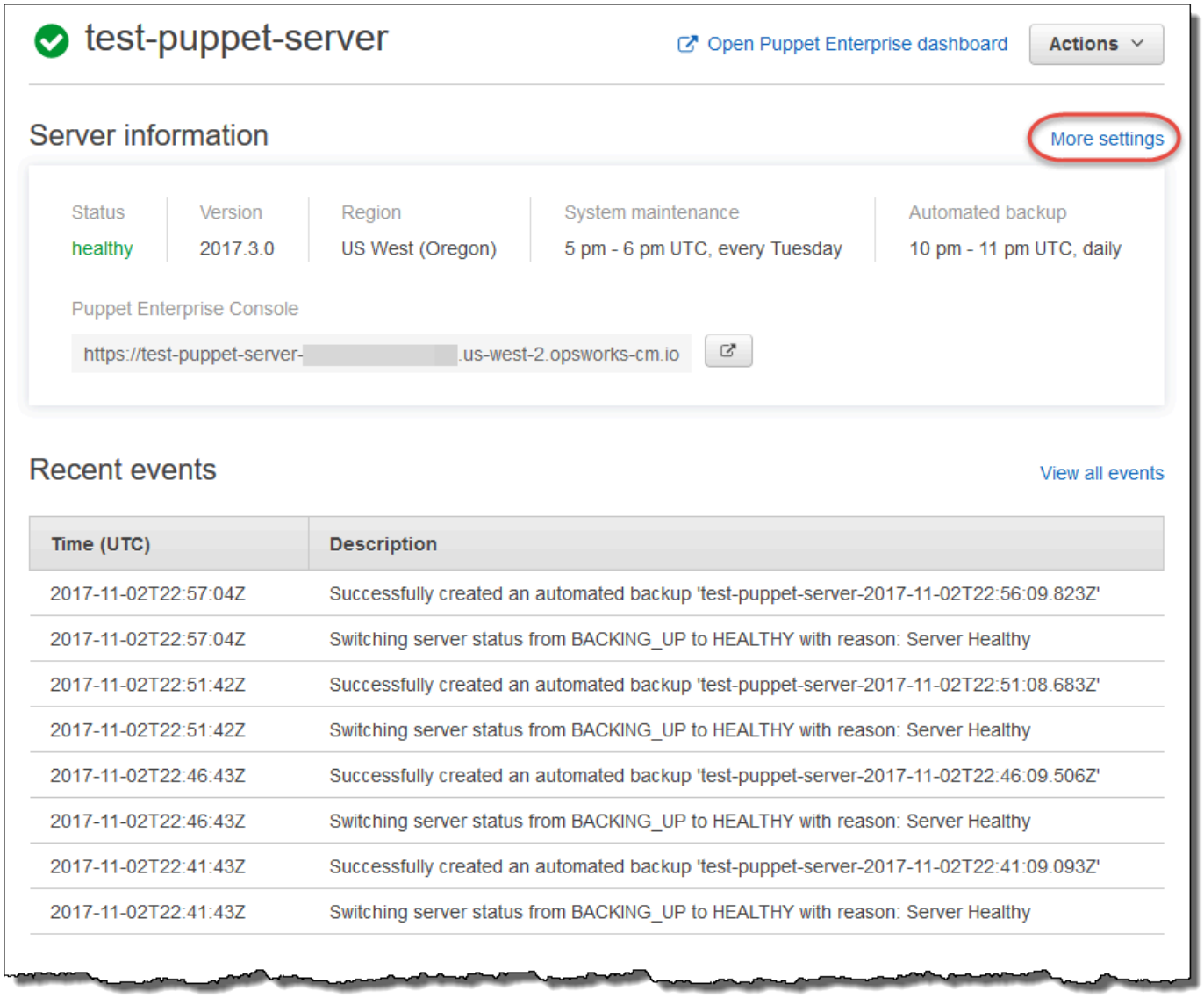
- Automated Backups
- Manual Backups
- Delete backups

Automated Backups

When you configure your OpsWorks for Puppet Enterprise server, you choose either automated or manual backups. OpsWorks for Puppet Enterprise starts automated backups during the hour and on the day that you choose in the **Automated backup** section of the **Configure advanced settings** page of the setup wizard. After your server is online, you can change backup settings by performing the following steps on the server's properties page.

To change automated backup settings

1. On the server's properties page, choose **More settings**.



2. To turn off automated backups, choose **No** for the **Enable automated backups** option. Save your changes; you do not need to go on to the next step.
3. In the **Automated Backup** section, change the frequency, start time, or generations to keep. Save your changes.

Manual Backups

You can start a manual backup at any time in the AWS Management Console, or by running the AWS CLI create-backup command. Manual backups are not included in the maximum 30 generations of automated backups that are stored. A maximum of 10 manual backups are stored, and must be manually deleted from Amazon S3.

To perform a manual backup in the AWS Management Console

1. On the **Puppet Enterprise servers** page, choose the server that you want to back up.
2. On the properties page for the server, in the left navigation pane, choose **Backups**.
3. Choose **Create backup**.
4. The manual backup is finished when the page shows a green check mark in the backup's **Status** column.

To perform a manual backup in the AWS CLI

You can add tags when you create a new, manual backup of an OpsWorks for Puppet Enterprise server. For more information about how to add tags when you create a manual backup, see Add Tags to a New Backup (CLI).

- To start a manual backup, run the following AWS CLI command.

```
aws opsworks-cm --region region name create-backup --server-name "Puppet server
 name" --description "optional descriptive string"
```

Delete backups

Deleting a backup permanently deletes it from the S3 bucket in which backups are stored.

To delete a backup in the AWS Management Console

1. On the **Puppet Enterprise servers** page, choose the server that you want to back up.
2. On the properties page for the server, in the left navigation pane, choose **Backups**.
3. Choose the backup that you want to delete, and then choose **Delete backup**. You can select only one backup at a time.
4. When you are prompted to confirm the deletion, fill the check box for **Delete the backup, which is stored in an S3 bucket**, and then choose **Yes, Delete**.

To delete a backup in the AWS CLI

- To delete a backup, run the following AWS CLI command, replacing the value of `--backup-id` with the ID of the backup that you want to delete. Backup IDs are in the format *ServerName-yyyyMMddHHmmssSSS*. For example, **puppet-server-20171218132604388**.

```
aws opsworks-cm --region region name delete-backup --backup-id ServerName-
yyyyMMddHHmmssSSS
```

Restore an OpsWorks for Puppet Enterprise Server from a Backup

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

After browsing through your available backups, you can easily choose a point in time from which to restore your OpsWorks for Puppet Enterprise server. Server backups contain configuration-management software persistent data such as modules, classes, node associations, database information (including reports, facts, etc.). Performing an in-place restoration of a server (that is, restoring the existing OpsWorks for Puppet Enterprise server to a new EC2 instance) reregisters nodes that were registered at the time of the backup that you use to restore the server, and switches traffic to the new instance if restoration is successful, and the restored OpsWorks for Puppet Enterprise server state is `Healthy`. Restoring to a newly-created OpsWorks for Puppet Enterprise server does not maintain node connections. Restoring a server does not update the version of Puppet software; it applies the same Puppet versions and configuration-management data that are available in the backup that you choose.

Restoring a server typically takes more time than creating a new server; the time depends on the size of the backup you choose. After restoration is complete, the old EC2 instance remains in a `Running` or `Stopped` state, but only temporarily. It is eventually terminated.

In this release, you can use the AWS CLI to restore a Puppet master in OpsWorks for Puppet Enterprise.

> **Note**
>
> You can also run the restore-server command to change the current instance type, or to restore or set your SSH key if it is lost or compromised.

To restore a server from a backup

1. In the AWS CLI, run the following command to return a list of available backups and their IDs. Make a note of the ID of the backup that you want to use. Backup IDs are in the format *myServerName-yyyyMMddHHmmssSSS*.

   ```
   aws opsworks-cm --region region name describe-backups
   ```

2. Run the following command.

   ```
   aws opsworks-cm --region region name restore-server --backup-id "myServerName-
   yyyyMMddHHmmssSSS" --instance-type "Type of instance" --key-pair "name of your EC2
    key pair" --server-name "name of Puppet master"
   ```

   The following is an example.

   ```
   aws opsworks-cm --region us-west-2 restore-server --backup-id
    "MyPuppetServer-20161120122143125" --server-name "MyPuppetServer"
   ```

3. Wait until restoration is complete.

System Maintenance in OpsWorks for Puppet Enterprise

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Mandatory system maintenance ensures that the latest AWS-tested versions of Puppet Server, including security updates, are always running on an OpsWorks for Puppet Enterprise server. System maintenance is required a minimum of once a week. By using the AWS CLI, you can configure daily automatic maintenance, if desired. You can also use the AWS CLI to perform system maintenance on demand, in addition to scheduled system maintenance.

When new versions of Puppet software become available, system maintenance is designed to update the version of Puppet Server on the server automatically, as soon as it passes AWS testing. AWS performs extensive testing to verify that Puppet upgrades are production-ready and do not disrupt existing customer environments, so there can be lags between Puppet software releases and their availability for application to existing OpsWorks for Puppet Enterprise servers. To update available versions of Puppet software on demand, see Starting system maintenance on demand in this topic.

System maintenance launches a new instance from a backup that is performed as part of the maintenance process, which helps reduce risk from degraded or impaired Amazon EC2 instances that undergo periodic maintenance.

> **Important**
>
> System maintenance deletes any files or custom configuration that you have added to the OpsWorks for Puppet Enterprise server. For more information about how to repair configuration or file loss, see Restoring custom configurations and files after maintenance in this topic.

Topics

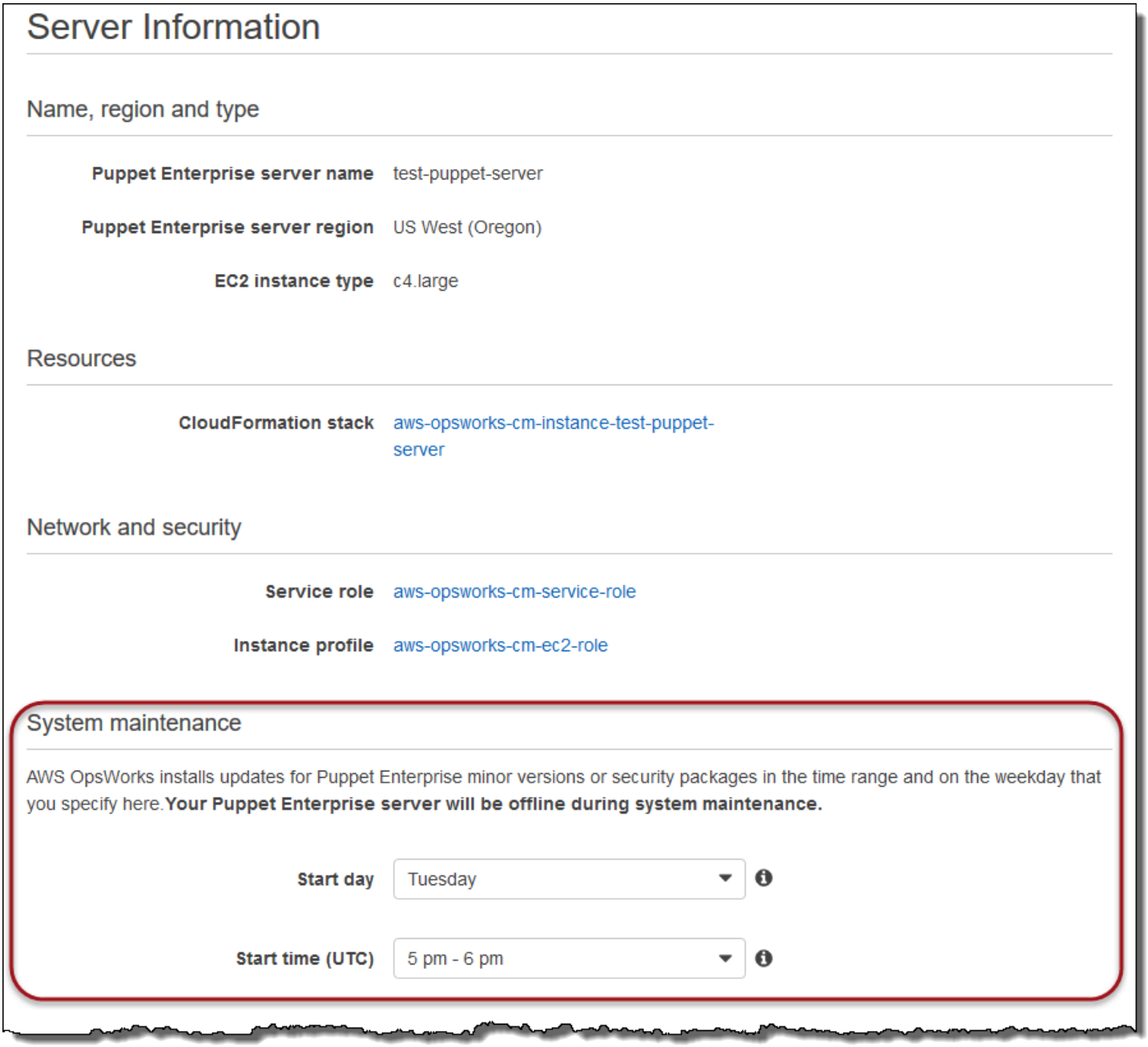
- Configuring system maintenance
- Starting system maintenance on demand
- Restoring custom configurations and files after maintenance

Configuring system maintenance

When you create a new OpsWorks for Puppet Enterprise server, you can configure a weekday and time, in Coordinated Universal Time (UTC), for system maintenance to start. Maintenance starts during the hour that you specify. Because you should expect the server to be offline during system

maintenance, choose a time of low server demand within regular office hours. The server status is `UNDER_MAINTENANCE` while maintenance is in progress.

You can also change the system maintenance settings on an existing OpsWorks for Puppet Enterprise server, by changing settings in the **System maintenance** area of the **Settings** page for your server, as shown in the following screenshot.



In the **System maintenance** section, set the day and hour that you want system maintenance to begin.

Configuring system maintenance by using the AWS CLI

You can also configure the system maintenance automatic start time by using the AWS CLI. The AWS CLI lets you configure daily automatic maintenance, if desired, by omitting the three-character weekday prefix.

In a `create-server` command, add the `--preferred-maintenance-window` parameter to your command, after specifying the requirements for creating the server instance (such as instance type, instance profile ARN, and service role ARN). In the following `create-server` example, `--preferred-maintenance-window` is set to `Mon:08:00`, meaning that you've set maintenance to start every Monday morning at 8:00 a.m. UTC.

```
aws opsworks-cm create-server --engine "Puppet" --engine-model "Monolithic"
 --engine-version "2017" --server-name "puppet-06" --instance-profile-arn
 "arn:aws:iam::1119001987000:instance-profile/aws-opsworks-cm-ec2-role"
 --instance-type "c4.large" --key-pair "amazon-test" --service-role-arn
 "arn:aws:iam::044726508045:role/aws-opsworks-cm-service-role" --preferred-maintenance-
window "Mon:08:00"
```

In an `update-server` command, you can update the `--preferred-maintenance-window` value alone, if desired. In the following example, the maintenance window is set to Friday night at 6:15 p.m. UTC.

```
aws opsworks-cm update-server --server-name "puppet-06" --preferred-maintenance-window
 "Fri:18:15"
```

To change the start time of the maintenance window to 6:15 p.m. UTC every day, omit the three-character weekday prefix, as shown in the following example.

```
aws opsworks-cm update-server --server-name "puppet-06" --preferred-maintenance-window
 "18:15"
```

For more information about setting the preferred system maintenance window by using the AWS CLI, see create-server and update-server.

Starting system maintenance on demand

To start system maintenance on demand, outside of your configured weekly or daily automatic maintenance, run the following AWS CLI command. You cannot start on-demand maintenance in the AWS Management Console.

```
aws opsworks-cm start-maintenance --server-name server_name
```

For more information about this command, see start-maintenance.

Restoring custom configurations and files after maintenance

System maintenance can delete or change custom files or configurations that you have added to your OpsWorks for Puppet Enterprise server.

If, after a maintenance run, your Puppet master is missing files or settings that you added by using RunCommand or SSH, you can use an Amazon Machine Image (AMI) to launch a new Amazon EC2 instance. AMIs are available that are built from a server's pre-maintenance configuration.

The new instance is in the same state that the Puppet master was before maintenance, and should include your missing files and settings.

> **Important**
>
> You cannot use the new instance to restore your server; the instance cannot be run as a Puppet master. You can use the instance only to recover your files and configuration settings.

To launch an EC2 instance from an AMI, in the Amazon EC2 console, open the **Launch** wizard, choose **My AMIs**, and then choose the AMI that has your server name. Follow Amazon EC2 wizard steps as you would for any other instance launch.

Adding Nodes Automatically in OpsWorks for Puppet Enterprise

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This topic describes how to add Amazon Elastic Compute Cloud (Amazon EC2) nodes to your OpsWorks for Puppet Enterprise server automatically. In Add Nodes for the Puppet Master to Manage, you learned how to use the `associate-node` command to add one node at a time to your Puppet Enterprise server. The code in this topic shows how to add nodes automatically using the unattended method. The recommended method of unattended (or automatic) association of new nodes is to configure the Amazon EC2 user data. By default, an OpsWorks for Puppet Enterprise server already has `puppet-agent` available for for Ubuntu, Amazon Linux, and RHEL node operating systems.

For information about how to disassociate a node, see Disassociate a Node from an OpsWorks for Puppet Enterprise Server in this guide, and `disassociate-node` in the OpsWorks for Puppet Enterprise API documentation.

Step 1: Create an IAM Role to Use as Your Instance Profile

Create an AWS Identity and Access Management (IAM) role to use as your EC2 instance profile, and attach the following policy to the IAM role. This policy allows the `opsworks-cm` API to communicate with the EC2 instance during node registration. For more information about instance profiles, see Using Instance Profiles in the Amazon EC2 documentation. For information about how to create an IAM role, see Creating an IAM Role in the Console in the Amazon EC2 documentation.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "opsworks-cm:AssociateNode",
                "opsworks-cm:DescribeNodeAssociationStatus",
                "opsworks-cm:DescribeServers",
                "ec2:DescribeTags"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

Step 2: Create Instances by Using an Unattended Association Script

To create EC2 instances, you can copy the user data script that is included in the Starter Kit to the `userdata` section of EC2 instance instructions, Amazon EC2 Auto Scaling group launch configurations, or an AWS CloudFormation template. The script is supported only for EC2 instances running Ubuntu and Amazon Linux operating systems. For more information about adding scripts to user data, see Running Commands on Your Linux Instance at Launch in the Amazon EC2 documentation. The easiest way to create a new node is to use the Amazon EC2 instance launch wizard. This walkthrough uses the Apache web server example module setup described in Getting Started with OpsWorks for Puppet Enterprise.

1. The user data script in the Starter Kit runs the `opsworks-cm` API `associate-node` command to associate a new node with your Puppet master. In this release, it also installs the current version of the AWS CLI on the node for you, in case it is not already running the most up-to-date version. Save this script to a convenient location as `userdata.sh`.

   By default, the name of the new registered node is the instance ID.
2. Follow the procedure in Launching an Instance in the EC2 documentation, with modifications here. In the EC2 instance launch wizard, choose an Amazon Linux AMI.
3. On the **Configure Instance Details** page, select **myPuppetinstanceprofile**, the role you created in Step 1: Create an IAM Role to Use as Your Instance Profile, as your IAM role.
4. In the **Advanced Details** area, upload the `userdata.sh` script that you created in Step 1.
5. No changes are needed on the **Add Storage** page. Go on to **Add Tags**.

   By applying tags to your EC2 instance, you can customize the behavior of `userdata.sh`. For this example, apply the role `apache_webserver` to your node by adding the following tag: **`pp_role`**, with the value **`apache_webserver`**.

   Setting the `pp_role` value on the node sets data values that are permanently stored in the node's agent certificate, enabling trusted classification of the node. For more information, see Extension requests (permanent certificate data) in the Puppet platform documentation.
6. On the **Configure Security Group** page, choose **Add Rule**, and then choose the type **HTTP** to open port 8080 for the Apache web server in this example.
7. Choose **Review and Launch**, and then choose **Launch**. When your new node starts, it applies the Apache configuration of the sample module you set up in Set Up the Starter Kit Apache Example.

8. When you open the webpage linked to the public DNS of your new node, you should see a website that is hosted by your Puppet-managed Apache web server.

Disassociate a Node from an OpsWorks for Puppet Enterprise Server

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section describes how to disassociate, or remove, a managed node from management by an OpsWorks for Puppet Enterprise server. This operation is performed on the command line or in the Puppet Enterprise console; you cannot disassociate nodes in the OpsWorks for Puppet Enterprise management console. Currently, the OpsWorks for Puppet Enterprise API does not allow for batch removal of multiple nodes. The command in this section disassociates one node at a time.

We recommend that you disassociate nodes from a Puppet master before you delete the server, so that the nodes continue to operate without trying to reconnect with the server. To do this, run the `disassociate-node` AWS CLI command. To completely remove a node from PE, you must disassociate the node and revoke its certificate, so that the node does not continuously attempt to check in with the Puppet master. You should also uninstall `puppet-agent` from nodes when you no longer want to manage them by using the Puppet master.

To disassociate nodes

1. In the AWS CLI, run the following command to disassociate nodes. *`Node_name`* is the name of the node that you want to disassociate; for Amazon EC2 instances, this is the instance ID.*`Server_name`* is the name of the Puppet master from which you want to disassociate the node. Both parameters are required. The `--region` parameter is not required unless you want to disassociate a node from a Puppet master that is not in your default region.

```
aws opsworks-cm --region Region_name disassociate-node --node-name Node_name --
server-name Server_name
```

The following command is an example.

```
aws opsworks-cm --region us-west-2 disassociate-node --node-name
 i-0010zzz00d66zzz90 --server-name opsworkstest
```

2. Wait until a response message indicates that the disassociation is finished.

For more information about how to delete an OpsWorks for Puppet Enterprise server, see Delete an OpsWorks for Puppet Enterprise Server.

See Also

- Remove nodes in the Puppet Enterprise documentation

Delete an OpsWorks for Puppet Enterprise Server

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section describes how to delete an OpsWorks for Puppet Enterprise server. Deleting a server also deletes its events, logs, and any modules that were stored on the server. Supporting resources (Amazon Elastic Compute Cloud instance, Amazon Elastic Block Store volume, etc.) are deleted also, along with all automated backups.

Although deleting a server does not delete nodes, they are no longer managed by the deleted server, and will continuously attempt to reconnect. For this reason, we recommend disassociating managed nodes before you delete a Puppet master. In this release, you can disassociate nodes by running an AWS CLI command.

Step 1: Disassociate Managed Nodes

Disassociate nodes from the Puppet master before you delete the server, so that the nodes continue to operate without trying to reconnect with the server. To do this, run the `disassociate-node` AWS CLI command.

To disassociate nodes

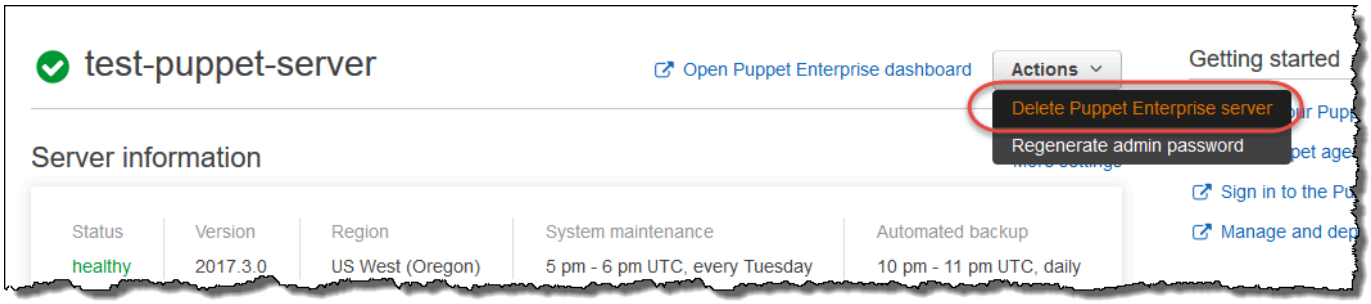
1. In the AWS CLI, run the following command to disassociate nodes. *`Server_name`* is the name of the Puppet master from which you want to disassociate the node. The value of `--node-name` can be an instance ID.

   ```
   aws opsworks-cm --region Region_name disassociate-node --node-name Node_name --
   server-name Server_name
   ```

2. Wait until a response message indicates that the disassociation is finished.

Step 2: Delete the Server

1. On the server's tile on the dashboard, expand the **Actions** menu.



2. Choose **Delete Puppet Enterprise server**.
3. When you are prompted to confirm the deletion, fill in the check box to delete associated roles and resources, and then choose **Yes, Delete**.

See Also

- Disassociate a Node from an OpsWorks for Puppet Enterprise Server

How to migrate an OpsWorks for Puppet Enterprise server to Amazon Elastic Compute Cloud (Amazon EC2)

> **Important**
>
> The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The instructions below describe how to migrate existing Puppet Enterprise servers to Amazon EC2, in case you want to continue using Puppet Enterprise for your configuration management needs outside of OpsWorks.

Topics

- Step 1: Contact Puppet to purchase a license
- Step 2: Get details about your OpsWorks for Puppet Enterprise server
- Step 3: Make a backup of your OpsWorks for Puppet Enterprise server
- Step 4: Launch a new EC2 instance
- Step 5: Install Puppet Enterprise on the new EC2 instance
- Step 6: Restore the backup on the new EC2 instance
- Step 7: Configure your Puppet license
- Step 8: Migrate your nodes
- Step 9: Delete your OpsWorks for Puppet Enterprise server

Step 1: Contact Puppet to purchase a license

When you migrate your servers to EC2, the new instance does not come with a Puppet license. To purchase a license key, follow the instructions on the Puppet website.

Step 2: Get details about your OpsWorks for Puppet Enterprise server

Find and save the values for your OpsWorks for Puppet Enterprise server.

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.

   Copy the name of the existing Amazon S3 bucket for your OpsWorks for Puppet Enterprise server. The bucket name has the format: `aws-opsworks-cm-`*`server-name`*`-`*`random-string`*

2. Run the **aws opsworks-cm describe-servers** command to get the configuration for your OpsWorks for Puppet Enterprise server.

   ```
   aws opsworks-cm describe-servers \
      --server-name server-name \
      --region region
   ```

   Store the values for `InstanceType`, `KeyPair`, `SubnetIds`, `SecurityGroupIds`, `InstanceProfileArn`, and `Endpoint` from the response.

3. Use SSH to connect to the existing OpsWorks for Puppet Enterprise server. You can use Session Manager in the EC2 console instead of SSH.

   Run the following command.

   ```
   rpm -qa | grep opsworks-cm-puppet-enterprise | cut -d '-' -f 5
   ```

   The response provides the Puppet Enterprise version (for example, 2019.8.10). Store this value.

You will use SSH or Session manager for the next step.

Step 3: Make a backup of your OpsWorks for Puppet Enterprise server

1. Run the following commands to make a local backup.

   ```
   mkdir /tmp/puppet-backup/
   sudo /opt/puppetlabs/bin/puppet-backup create --dir=/tmp/puppet-backup/
   ```

2. Run the following command to store the name for the backup.

   ```
   ls /tmp/puppet-backup/
   PUPPET_BACKUP=$(ls /tmp/puppet-backup/)
   ```

3. Run the following command to upload your backup to an S3 bucket. Replace *S3-Bucket* with the value from step 1 in Step 2: Get details about your OpsWorks for Puppet Enterprise server.

   ```
   aws s3 cp /tmp/puppet-backup/PUPPET_BACKUP s3://S3_Bucket/tmp/puppet-backup/
   ```

   Store the PUPPET_BACKUP and S3_BUCKET values. You will import those values to the new EC2 instance.

You can exit the SSH or Session Manager session.

Step 4: Launch a new EC2 instance

Launch a new EC2 instance from the EC2 console at https://console.aws.amazon.com/ec2/ using the same configuration as the OpsWorks for Puppet Enterprise server.

Parameter name	Value
OS	Amazon Linux 2
Instance type	The `InstanceType` value from step 2 of Step 2: Get details about your OpsWorks for Puppet Enterprise server.
Key pair name	The `KeyPair` value from step 2 of Step 2: Get details about your OpsWorks for Puppet Enterprise server.
VPC	The VPC of the `SubnetIds` from step 2 of Step 2: Get details about your OpsWorks for Puppet Enterprise server.
Subnet	The `SubnetIds` from step 2 of Step 2: Get details about your OpsWorks for Puppet Enterprise server.
Select existing security group -> Common security groups	The `SecurityGroupIds` from step 2 of Step 2: Get details about your OpsWorks for Puppet Enterprise server.
Storage	At least 120 GB.
IAM instance profile	The `InstanceProfileArn` from step 2 of Step 2: Get details about your OpsWorks for Puppet Enterprise server.

If you want to create and attach an Elastic IP to the new instance, copy the instance ID of the new instance, and complete the steps in (Optional) Step 4.1: Create and attach an Elastic IP.

(Optional) Step 4.1: Create and attach an Elastic IP

By using an Elastic IP address, you can mask the failure of an instance or software by rapidly remapping the address to another instance in your account.

To create and associate an Elastic IP address

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose **Elastic IPs**.
3. Choose **Allocate Elastic IP address**.
4. From the **Allocate Elastic IP address** page, choose **Allocate**. This creates a Public IPv4 address.
5. Copy the **Allocated IPv4 address**.
6. From **Actions**, choose **Associate Elastic IP address**.
7. For **Instance**, enter the instance ID for the new instance.
8. Choose **Associate**.

Step 5: Install Puppet Enterprise on the new EC2 instance

Use SSH to connect to the new EC2 instance. You can use Session Manager in the EC2 console instead of SSH.

```
# switch to sudo user
sudo -i

# Setup environment variables
PUPPET_ENTERPRISE_VERSION=Puppet Enterprise version from step 2.3
hostname Public IPv4 DNS or Custom Domain if available

# Install Puppet Enterprise
curl -JLO https://pm.puppetlabs.com/puppet-enterprise/$PUPPET_ENTERPRISE_VERSION/
puppet-enterprise-$PUPPET_ENTERPRISE_VERSION-el-7-x86_64.tar.gz
tar -xf puppet-enterprise-$PUPPET_ENTERPRISE_VERSION-el-7-x86_64.tar.gz

./puppet-enterprise-$PUPPET_ENTERPRISE_VERSION-el-7-x86_64/puppet-enterprise-installer
```

You can keep your SSH or Session Manager session open for the next step.

Step 6: Restore the backup on the new EC2 instance

```
# Setup environment variables
S3_BUCKET=S3 bucket name from step 2.1
PUPPET_BACKUP=Puppet backup file name from step 3.2

# download backup
aws s3 cp s3://$S3_BUCKET/tmp/puppet-backup/$PUPPET_BACKUP

# Prepare Puppet Enterprise backup to remove OpsWorks metadata
mkdir output
tar -xf $PUPPET_BACKUP -C output/
cd output/
rm -f opt/puppetlabs/facter/facts.d/opsworks.json
tar -cf ../$PUPPET_BACKUP *
cd ..
rm -rf output/

# Restore from backup
PATH=$PATH:/opt/puppetlabs/puppet/bin/
puppet-backup restore $PUPPET_BACKUP
puppet agent -t
```

You can access the Puppet console for the restored EC2 instance at https://*Public IPv4 of the instance*. You can find the Public IPv4 DNS on the instance's details page in the EC2 console. The login credentials are the same credentials you use to access your OpsWorks for Puppet Enterprise server.

You can keep your SSH or Session Manager session open for the next step.

Step 7: Configure your Puppet license

Follow the steps on the Puppet website to configure your license.

You can keep your SSH or Session Manager session open for the next step.

Step 8: Migrate your nodes

There are two types of domains supported by the OpsWorks for Puppet Enterprise servers:

- BYODC (Bring Your Own Domain and Certificate)
- OpsWorks endpoint

Step 8.1: For BYODC (Bring Your Own Domain and Certificate)

For these nodes, all you need to do is point the Custom Domain in your DNS provider to the Public IPv4 DNS or Public IPv4 address of the new EC2 instance.

Step 8.2: For OpsWorks endpoint

For an OpsWorks endpoint, the Puppet documentation recommends to uninstall the Puppet agent on the node and then install the Puppet agent using the newly restored Puppet Enterprise server.

> **Note**
>
> While Puppet doesn't have an automated procedure to move an agent node, there are a few modules that Puppet community members have published on the Puppet Forge website to accomplish automated node migration. These modules include the `pe_migrate` module and a second migrate module by a different author. The modules on the Puppet Forge website **are not supported by Puppet or OpsWorks** unless explicitly noted within the Forge module. We recommend using caution with these modules and testing them before using widely.

The following sections provide the steps to uninstall and reinstall Puppet agents on Linux instances.

Topics

- Step 8.2.1: Copy the uninstaller from the Puppet server
- Step 8.2.2: Download the uninstaller and run it on a node
- Step 8.2.3: Reinstall the Puppet agent on a node

Step 8.2.1: Copy the uninstaller from the Puppet server

Before you uninstall the agent, be sure the node's IAM instance profile provides S3 ReadOnly permissions.

Run the following command to copy the uninstaller from the Puppet server to the S3 bucket.

```
aws s3 cp \
   /opt/puppetlabs/bin/puppet-enterprise-uninstaller \
   s3://$S3_BUCKET/tmp/puppet-enterprise-uninstaller
```

After running the command, you can log out of the Puppet server's SSH or Session Manager session.

Step 8.2.2: Download the uninstaller and run it on a node

Use SSH to connect to the node. You can use Session Manager in the EC2 console instead of SSH if the node is an EC2 instance.

```
sudo -i

S3_BUCKET=aws-opsworks-cm-abcdefg-uuhtyn6messn
aws s3 cp s3://$S3_BUCKET/tmp/puppet-enterprise-uninstaller /opt/puppetlabs/bin/
chmod 700 /opt/puppetlabs/bin/puppet-enterprise-uninstaller
/opt/puppetlabs/bin/puppet-enterprise-uninstaller
```

You can keep your SSH or Session Manager session open for the next step.

Step 8.2.3: Reinstall the Puppet agent on a node

Complete the following steps to reinstall the Puppet agent on a node.

Topics

- Step 8.2.3.1: Install the Puppet agent with the correct configuration
- Step 8.2.3.2: Accept the certificate in the Puppet console
- Step 8.2.3.3: Check the node into the Puppet Enterprise server

Step 8.2.3.1: Install the Puppet agent with the correct configuration

Run the following command to install the Puppet agent.

```
curl -k https://Public_IPv4_DNS:8140/packages/current/install.bash | bash
```

You can keep your SSH or Session Manager session open for step 8.2.2.3.

Step 8.2.3.2: Accept the certificate in the Puppet console

1. Go to the Puppet server's console at `https://`*`Public_IPv4_DNS`*.
2. Choose **Certificates**, and then **Unsigned certificates**.
3. Choose **Accept** to sign the Puppet agent's certificate.

Step 8.2.3.3: Check the node into the Puppet Enterprise server

Run the following command on the node to check it into the server.

```
puppet agent -t
```

The node should now be visible in the Puppet server's console.

Step 9: Delete your OpsWorks for Puppet Enterprise server

You can use either the OpsWorks console or AWS CLI to delete your OpsWorks for Puppet Enterprise server.

To delete your server using the OpsWorks console

1. Sign in to the AWS Management Console and open the OpsWorks console at https://console.aws.amazon.com/opsworks/.
2. Choose **Puppet Enterprise servers** from the navigation pane.
3. On the **Puppet Enterprise servers** page, choose the server you want to delete.
4. From **Actions**, choose **Delete Puppet Enterprise server**.

To delete your server using the AWS CLI

Run the following command.

```
aws opsworks-cm delete-server \
   --server-name server-name \
   --region region
```

Logging OpsWorks for Puppet Enterprise API Calls with AWS CloudTrail

Important

The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks for Puppet Enterprise is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in OpsWorks for Puppet Enterprise. CloudTrail captures all API calls for OpsWorks for Puppet Enterprise as events, including calls from the OpsWorks for Puppet Enterprise console and from code calls to the OpsWorks for Puppet Enterprise APIs. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for OpsWorks for Puppet Enterprise. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to OpsWorks for Puppet Enterprise, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

OpsWorks for Puppet Enterprise Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in OpsWorks for Puppet Enterprise, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for OpsWorks for Puppet Enterprise, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all regions. The trail logs events from all regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All OpsWorks for Puppet Enterprise actions are logged by CloudTrail and are documented in the OpsWorks for Puppet Enterprise API Reference. For example, calls to the **CreateServer**, **CreateBackup**, and **DescribeServers** actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understanding OpsWorks for Puppet Enterprise Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following example shows a CloudTrail log entry for the OpsWorks for Puppet Enterprise `CreateServer` action.

```
{"eventVersion":"1.05",
"userIdentity":{
    "type":"AssumedRole",
    "principalId":"ID number:OpsWorksCMUser",
    "arn":"arn:aws:sts::831000000000:assumed-role/Admin/OpsWorksCMUser",
    "accountId":"831000000000","accessKeyId":"ID number",
    "sessionContext":{
        "attributes":{
```

```
            "mfaAuthenticated":"false",
            "creationDate":"2017-01-05T22:03:47Z"
            },
        "sessionIssuer":{
            "type":"Role",
            "principalId":"ID number",
            "arn":"arn:aws:iam::831000000000:role/Admin",
            "accountId":"831000000000",
            "userName":"Admin"
            }
        }
    },
"eventTime":"2017-01-05T22:18:23Z",
"eventSource":"opsworks-cm.amazonaws.com",
"eventName":"CreateServer",
"awsRegion":"us-west-2",
"sourceIPAddress":"101.25.190.51",
"userAgent":"console.amazonaws.com",
"requestParameters":{
    "serverName":"test-puppet-server",
    "engineModel":"Single",
    "engine":"Puppet",
    "instanceProfileArn":"arn:aws:iam::831000000000:instance-profile/aws-opsworks-cm-
ec2-role",
    "backupRetentionCount":3,"serviceRoleArn":"arn:aws:iam::831000000000:role/service-
role/aws-opsworks-cm-service-role",
    "engineVersion":"12",
    "preferredMaintenanceWindow":"Fri:21:00",
    "instanceType":"t2.medium",
    "subnetIds":["subnet-1e111f11"],
    "preferredBackupWindow":"Wed:08:00"
    },
"responseElements":{
    "server":{
        "endpoint":"test-puppet-server-xxxx8u4390xo6pd9.us-west-2.opsworks-cm.io",
        "createdAt":"Jan 5, 2017 10:18:22 PM",
        "serviceRoleArn":"arn:aws:iam::831000000000:role/service-role/aws-opsworks-cm-
service-role",
        "preferredBackupWindow":"Wed:08:00",
        "status":"CREATING",
        "subnetIds":["subnet-1e111f11"],
        "engine":"Puppet",
        "instanceType":"t2.medium",
        "serverName":"test-puppet-server",
```

```
        "serverArn":"arn:aws:opsworks-cm:us-west-2:831000000000:server/test-puppet-
server/8ezz7f6z-e91f-4z10-89z5-8c6219zzz09f",
        "engineModel":"Single",
        "backupRetentionCount":3,
        "engineAttributes":[
            {"name":"PUPPET_ADMIN_PASSWORD","value":"*** Redacted ***"},
            {"name":"PUPPET_API_CA_CERT","value":"*** Redacted ***"},
            ],
        "engineVersion":"12.11.1",
        "instanceProfileArn":"arn:aws:iam::831000000000:instance-profile/aws-opsworks-
cm-ec2-role",
        "preferredMaintenanceWindow":"Fri:21:00"
        }
    },
"requestID":"de7z64z9-d394-12ug-8081-7zz0386fbcb6",
"eventID":"8z7z18dz-6z90-47bz-87cf-e8346428zzz3",
"eventType":"AwsApiCall",
"recipientAccountId":"831000000000"
}
```


Troubleshooting OpsWorks for Puppet Enterprise

Important

The AWS OpsWorks for Puppet Enterprise service reached end of life on March 31, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This topic contains some common OpsWorks for Puppet Enterprise issues, and suggested solutions for those issues.

Topics

- General troubleshooting tips
- Troubleshooting specific errors
- Additional help and support

General troubleshooting tips

If you are unable to create or work with a Puppet master, you can view error messages or logs to help you troubleshoot the issue. The following tasks describe general places to start when you are troubleshooting a Puppet master issue. For information about specific errors and solutions, see the Troubleshooting specific errors section of this topic.

- Use the OpsWorks for Puppet Enterprise console to view error messages if a Puppet master fails to start. On the Puppet master properties page, error messages related to launching and running the server are shown at the top of the page. Errors can come from OpsWorks for Puppet Enterprise, AWS CloudFormation, or Amazon EC2, services that are used to create a Puppet master. On the properties page, you can also view events that occur on a running server, which can contain failure event messages.
- To help resolve EC2 issues, connect to your server's instance by using SSH, and view logs. EC2 instance logs are stored in the `/var/log/aws/opsworks-cm` directory. These logs capture command outputs while OpsWorks for Puppet Enterprise launches a Puppet master.

Troubleshooting specific errors

Topics

- Server is in a Connection lost state
- Server creation fails with "requested configuration is currently not supported" message
- Unable to create the server's Amazon EC2 instance
- Service role error prevents server creation
- Elastic IP address limit exceeded
- Unattended node association fails
- System maintenance fails

Server is in a Connection lost state

Problem: A server's status shows as **Connection lost**.

Cause: This most commonly occurs when an entity outside of OpsWorks makes changes to an OpsWorks for Puppet Enterprise server or its supporting resources. OpsWorks cannot connect to Puppet Enterprise servers in **Connection lost** states to handle maintenance tasks such as creating

backups, applying operating system patches, or updating Puppet. As a result, your server might be missing important updates, susceptible to security issues, or otherwise not operating as expected.

Solution: Try the following steps to restore the server's connection.

1. Be sure that your service role has all required permissions.

   a. On the **Settings** page for your server, in **Network and security**, choose the link for the service role that the server is using. This opens the service role for viewing in the IAM console.

   b. On the **Permissions** tab, verify that `AWSOpsWorksCMServiceRole` is in the **Permissions policies** list. If it isn't listed, add the `AWSOpsWorksCMServiceRole` managed policy manually to the role.

   c. On the **Trust relationships** tab, verify that the service role has a trust policy that trusts the `opsworks-cm.amazonaws.com` service to assume roles on your behalf. For more information about how to use trust policies with roles, see Modifying a role (console), or the AWS Security Blog post, How to use trust policies with IAM roles.

2. Be sure that your instance profile has all required permissions.

   a. On the **Settings** page for your server, in **Network and security**, choose the link for the instance profile that the server is using. This opens the instance profile for viewing in the IAM console.

   b. On the **Permissions** tab, verify that `AmazonEC2RoleforSSM` and `AWSOpsWorksCMInstanceProfileRole` are both in the **Permissions policies** list. If one or both aren't listed, add these managed policies manually to the role.

   c. On the **Trust relationships** tab, verify that the service role has a trust policy that trusts the `ec2.amazonaws.com` service to assume roles on your behalf. For more information about how to use trust policies with roles, see Modifying a role (console), or the AWS Security Blog post, How to use trust policies with IAM roles.

3. In the Amazon EC2 console, be sure that you are in the same region as the region of the OpsWorks for Puppet Enterprise server, and then restart the EC2 instance that your server is using.

   a. Choose the EC2 instance that is named `aws-opsworks-cm-instance-`*`server-name`*.

   b. On the **Instance state** menu, choose **Reboot instance**.

   c. Allow up to 15 minutes for your server to restart and be fully online.

4. In the OpsWorks for Puppet Enterprise console, on the server details page, verify that the server status is now **healthy**.

If the server status is still **Connection lost** after performing the preceding steps, try one of the following.

- Replace the server by creating a new one and deleting the original. If data on the current server is important to you, restore the server from a recent backup, and verify the data is up to date before deleting the original, unresponsive server.
- Contact AWS support.

Server creation fails with "requested configuration is currently not supported" message

Problem: You are trying to create a Puppet Enterprise server, but server creation fails with an error message that is similar to "The requested configuration is currently not supported. Please check the documentation for supported configurations."

Cause: An unsupported instance type might have been specified for the Puppet master. If you choose to create the Puppet server in a VPC that has a non-default tenancy, such as one for dedicated instances, all instances inside the specified VPC must also be of dedicated or host tenancy. Because some instance types, such as t2, are available only with default tenancy, the Puppet master instance type might not be supportable by the specified VPC, and server creation fails.

Solution: If you choose a VPC that has a non-default tenancy, use an m4 instance type, which can support dedicated tenancy.

Unable to create the server's Amazon EC2 instance

Problem: Server creation failed with an error message similar to the following: "The following resource(s) failed to create: [EC2Instance]. Failed to receive 1 resource signal(s) within the specified duration."

Cause: This is most likely because the EC2 instance doesn't have network access.

Solution: Ensure the instance has outbound Internet access, and the AWS service agent is able to issue commands. Be sure that your VPC (a VPC with a single public subnet) has **DNS resolution** enabled, and that your subnet has the **Auto-assign Public IP** setting enabled.

Service role error prevents server creation

Problem: Server creation fails with an error message that states, "Not authorized to perform sts:AssumeRole."

Cause: This can occur when the service role you are using lacks adequate permissions to create a new server.

Solution: Open the OpsWorks for Puppet Enterprise console; use the console to generate a new service role and an instance profile role. If you would prefer to use your own service role, attach the **AWSOpsWorksCMServiceRole** policy to the role. Verify that **opsworks-cm.amazonaws.com** is listed among services in the role's **Trust relationships**. Verify that the service role that is associated with the Puppet master has the **AWSOpsWorksCMServiceRole** managed policy attached.

Elastic IP address limit exceeded

Problem: Server creation fails with an error message that states, "The following resource(s) failed to create: [EIP, EC2Instance]. Resource creation cancelled, the maximum number of addresses has been reached."

Cause: This occurs when your account has used the maximum number of Elastic IP (EIP) addresses. The default EIP address limit is five.

Solution: You can either release existing EIP addresses or delete ones that your account is not actively using, or you can contact AWS Customer Support to increase the limit of EIP addresses that is associated with your account.

Unattended node association fails

Problem: Unattended, or automatic, association of new Amazon EC2 nodes is failing. Nodes that should have been added to the Puppet master are not showing up in the Puppet Enterprise dashboard.

Cause: This can occur when you do not have an IAM role set up as an instance profile that permits `opsworks-cm` API calls to communicate with new EC2 instances.

Solution: Attach a policy to your EC2 instance profile that allows the `AssociateNode` and `DescribeNodeAssociationStatus` API calls to work with EC2, as described in Adding Nodes Automatically in OpsWorks for Puppet Enterprise.

System maintenance fails

AWS OpsWorks CM performs weekly system maintenance to ensure that the latest AWS-tested versions of Puppet Server, including security updates, are always running on an OpsWorks for Puppet Enterprise server. If, for any reason, system maintenance fails, AWS OpsWorks CM notifies you of the failure. For more information about system maintenance, see System Maintenance in OpsWorks for Puppet Enterprise.

This section describes possible reasons for failure and suggests solutions.

Topics

- Service role or instance profile error prevents system maintenance

Service role or instance profile error prevents system maintenance

Problem: System maintenance fails with an error message that states, "Not authorized to perform sts:AssumeRole", or a similar error message about permissions.

Cause: This can occur when either the service role or instance profile you are using lacks adequate permissions to perform system maintenance on the server.

Solution: Be sure that your service role and instance profile have all required permissions.

1. Be sure that your service role has all required permissions.
    a. On the **Settings** page for your server, in **Network and security**, choose the link for the service role that the server is using. This opens the service role for viewing in the IAM console.
    b. On the **Permissions** tab, verify that `AWSOpsWorksCMServiceRole` is attached to the service role. If `AWSOpsWorksCMServiceRole` is not listed, add this policy to the role.
    c. Verify that **opsworks-cm.amazonaws.com** is listed among services in the role's **Trust relationships**. For more information about how to use trust policies with roles, see Modifying a role (console), or the AWS Security Blog post, How to use trust policies with IAM roles.
2. Be sure that your instance profile has all required permissions.
    a. On the **Settings** page for your server, in **Network and security**, choose the link for the instance profile that the server is using. This opens the instance profile for viewing in the IAM console.

b. On the **Permissions** tab, verify that `AmazonEC2RoleforSSM` and `AWSOpsWorksCMInstanceProfileRole` are both in the **Permissions policies** list. If one or both aren't listed, add these managed policies manually to the role.

c. On the **Trust relationships** tab, verify that the service role has a trust policy that trusts the `ec2.amazonaws.com` service to assume roles on your behalf. For more information about how to use trust policies with roles, see Modifying a role (console), or the AWS Security Blog post, How to use trust policies with IAM roles.

Additional help and support

If you do not see your specific problem described in this topic, or you have tried the suggestions in this topic and are still having problems, visit the OpsWorks forums.

You can also visit the AWS Support Center. The AWS Support Center is the hub for creating and managing AWS Support cases. The AWS Support Center also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.

AWS OpsWorks for Chef Automate

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

AWS OpsWorks for Chef Automate lets you run a Chef Automate server in AWS. You can provision a Chef server within minutes, and let AWS OpsWorks for Chef Automate handle its operations, backups, restorations, and software upgrades. AWS OpsWorks for Chef Automate frees you to focus on core configuration management tasks, instead of managing a Chef server.

A Chef Automate server manages the configuration of nodes in your environment by instructing `chef-client` which Chef recipes to run on the nodes, stores information about nodes, and serves as a central repository for your Chef cookbooks. AWS OpsWorks for Chef Automate provides Chef servers that include premium features of Chef Automate: Chef Infra and Chef InSpec.

An AWS OpsWorks for Chef Automate server runs on an Amazon Elastic Compute Cloud instance. AWS OpsWorks for Chef Automate servers are configured to run the newest version of Amazon Linux (Amazon Linux 2). For information about what has changed in this version of Chef Automate, see the Chef Automate Release Notes. The following table describes the Chef components that are installed on an AWS OpsWorks for Chef Automate server.

Component name	Description	Version installed on AWS OpsWorks for Chef Automate server
Chef Automate	Chef Automate is an enterprise server software package that provides automated workflow for continuous deployment, and insights about managed nodes in a web-based	2.0

Component name	Description	Version installed on AWS OpsWorks for Chef Automate server
	management console. Chef Automate delivers infrastructure automatio n by including Chef Infra, security and compliance information and enforceme nt by including Chef InSpec, and automated deployment by including Chef Habitat. For more information about Chef Automate, see Chef Automate on the Chef website.	
Chef Infra	Formerly called Chef Server, Chef Infra Server uses the Chef Infra Client (`chef-clie nt`) agent to continuou sly apply configurations to managed nodes to maintain a desired state. For more information about Infra, see Chef Infra on the Chef website.	12.x

Component name	Description	Version installed on AWS OpsWorks for Chef Automate server
Chef InSpec	Chef InSpec describes security and compliance rules that can be shared between software engineers, operations, and security engineers. Compliance, security, and other policy requirements form the framework for automated tests that the `chef-client` agent can run against managed nodes, ensuring consistent enforcement of standards. For more information about InSpec, see Chef InSpec on the Chef website.	3.9.0

The minimum supported version of `chef-client` on nodes associated with an AWS OpsWorks for Chef Automate server is 13.*x*. We recommend running at least 14.10.9, or the most current, stable `chef-client` version.

When new minor versions of Chef software become available, system maintenance is designed to update the minor version of Chef Automate and Chef Server on the server automatically, as soon as it passes AWS testing. AWS performs extensive testing to verify that Chef upgrades are production-ready and do not disrupt existing customer environments, so there can be lags between Chef software releases and their availability for application to existing OpsWorks for Chef Automate servers. System maintenance also upgrades your server to the newest version of Amazon Linux.

You can connect any on-premises computer or EC2 instance that is running a supported operating system and has network access to an AWS OpsWorks for Chef Automate server. For a list of

supported operating systems for nodes that you want to manage, see the Chef website. The `chef-client` agent software is installed on nodes that you want to manage with a Chef server.

Topics

- Region Support for AWS OpsWorks for Chef Automate
- AWS OpsWorks for Chef Automate End of Life FAQs
- Upgrade an AWS OpsWorks for Chef Automate Server to Chef Automate 2
- Getting Started with AWS OpsWorks for Chef Automate
- Create an AWS OpsWorks for Chef Automate Server by using AWS CloudFormation
- Update an AWS OpsWorks for Chef Automate Server to Use a Custom Domain
- Regenerate the starter kit for an AWS OpsWorks for Chef Automate server
- Working with Tags on AWS OpsWorks for Chef Automate Resources
- Back Up and Restore an AWS OpsWorks for Chef Automate Server
- System Maintenance in AWS OpsWorks for Chef Automate
- Compliance Scans in AWS OpsWorks for Chef Automate
- Disassociate a Node from an AWS OpsWorks for Chef Automate Server
- Delete an AWS OpsWorks for Chef Automate Server
- Reset Chef Automate Dashboard Credentials
- Logging AWS OpsWorks for Chef Automate API Calls with AWS CloudTrail
- Troubleshooting AWS OpsWorks for Chef Automate

Region Support for AWS OpsWorks for Chef Automate

The following regional endpoints support AWS OpsWorks for Chef Automate servers. AWS OpsWorks for Chef Automate creates resources that are associated with your Chef servers, such as instance profiles, users, and service roles, in the same regional endpoint as your Chef server. Your Chef server must be in a VPC. You can use a VPC that you create or already have, or use the default VPC.

- US East (Ohio) Region
- US East (N. Virginia) Region
- US West (N. California) Region

- US West (Oregon) Region
- Asia Pacific (Tokyo) Region
- Asia Pacific (Singapore) Region
- Asia Pacific (Sydney) Region
- Europe (Frankfurt) Region
- Europe (Ireland) Region

AWS OpsWorks for Chef Automate End of Life FAQs

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution.

Topics

- How will existing users be affected by this End of Life?
- What happens to my servers if I don't take any action?
- What alternatives can I transition to?
- Is the service still accepting new customers?
- Will the End of Life affect all AWS Regions at the same time?
- What level of technical support is available?
- I am a current customer of OpsWorks for Chef Automate and I need to launch a server in an account which was not using the service previously. Am I able to do this?
- Will there be any major feature releases over the next year?

How will existing users be affected by this End of Life?

Existing customers will be unaffected until May 5, 2024, the End of Life date for OpsWorks for Chef Automate. After the End of Life date, customers will no longer be able to manage their servers using the OpsWorks console or API.

What happens to my servers if I don't take any action?

Starting May 5, 2024, you will no longer be able to manage your servers using the OpsWorks console or API. At that time, we will stop performing any ongoing management functions for your servers such as backups or maintenance. To limit the impact to customers, we will leave any EC2 instances running that are backing up Chef Automate servers, but their licenses will no longer be valid as usage is no longer covered (or billed for) under the OpsWorks for Chef Automate service agreement with Chef. You will need to contact Chef to obtain a new license. When you contact Chef, be sure to tell them you are an existing OpsWorks for Chef Automate customer and you are transitioning from OpsWorks.

What alternatives can I transition to?

AWS and Progress Chef recommend you migrate to their new Chef SaaS offering so that you can continue to benefit from a fully-managed Chef Automate service. To get started with Chef SaaS, you can contact Chef to obtain documentation about how to setup a Chef SaaS account and transition your data and nodes.

If Chef SaaS will not meet your needs because you prefer to run Chef Automate on EC2 instances in AWS accounts you control, Chef provides multiple options including an AWS Marketplace Bring Your Own License (BYOL) model and self-hosting on EC2. You can contact Progress Chef for more information about how to execute such a transition.

Is the service still accepting new customers?

No. AWS OpsWorks for Chef Automate is no longer accepting new customers.

Will the End of Life affect all AWS Regions at the same time?

Yes. The API and Console will reach End of Life and be unusable as of May 5, 2024 in all AWS Regions. For information about the AWS Regions where AWS OpsWorks for Chef Automate is available, see AWS Regional Services List.

What level of technical support is available?

AWS will continue to provide the same level of support for OpsWorks for Chef Automate that customers have today up until the End of Life date. If you have questions or concerns, you can contact the AWS Support Team on AWS re:Post or through AWS Premium Support. For transition support, we recommend that customers contact Progress Chef.

I am a current customer of OpsWorks for Chef Automate and I need to launch a server in an account which was not using the service previously. Am I able to do this?

Generally not, unless there are exceptional circumstances to do so. If you have a special situation, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support with the details and justification for this and we will review your request.

Will there be any major feature releases over the next year?

No. As the service is reaching End of Life, we will not release any new features. However, we will continue to make security improvements and manage servers as expected until the End of Life date.

Upgrade an AWS OpsWorks for Chef Automate Server to Chef Automate 2

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Prerequisites for Upgrading to Chef Automate 2

Before you get started, be sure you understand the new features that Chef Automate 2 adds, and features that Chef Automate 2 does not support. For information about the new and unsupported features in Chef Automate 2, see the Chef Automate 2 documentation on the Chef website.

A server running Chef Automate 1 must have had at least one successful maintenance run after November 1, 2019 to be eligible for upgrade.

As with any maintenance operation on your AWS OpsWorks for Chef Automate server, the server is offline during the upgrade. You should plan for up to three hours of downtime during the upgrade process.

You need the sign-in credentials for this server for the Chef Automate dashboard website. When the upgrade is finished, you should sign in to the Chef Automate dashboard and verify that your nodes and configuration information are not changed.

> **Important**
>
> When you are ready to upgrade your AWS OpsWorks for Chef Automate server to Chef Automate 2, use only the instructions here to upgrade. Because AWS OpsWorks for Chef Automate automates many of the upgrade processes, such as backup creation, do not follow upgrade instructions on the Chef website.

About the Upgrade Process

During the upgrade process, your server is backed up before starting upgrade and after finishing the upgrade. The following backups are created:

- A backup of the server when it's still running Chef Automate 1 (version 12.17.33).
- A backup of the server after upgrade is finished and the server is running Chef Automate 2 (version 2019-08).

The upgrade process terminates the Amazon EC2 instance that the server was using when it ran Chef Automate 1. A new instance is created to run the Chef Automate 2 server.

Upgrade to Chef Automate 2 (Console)

1. Sign in to the AWS Management Console and open the OpsWorks console at https://console.aws.amazon.com/opsworks/.
2. In the left navigation pane, choose **AWS OpsWorks for Chef Automate**.
3. Choose a server to view its properties page. A blue banner at the top of the page should indicate whether the server is eligible for upgrade to Chef Automate 2.

   > **Note**
   >
   > A server running Chef Automate 1 must have had at least one successful maintenance run after November 1, 2019 to be eligible for upgrade.

4. If the server is eligible for upgrade, choose **Start upgrade**.
5. Allow up to three hours for upgrade. During the upgrade process, the properties page displays server status as **Under maintenance**.
6. When the upgrade is finished, the properties page displays the following two messages: **Successfully upgraded to Automate 2** and **Maintenance completed successfully**. The server status should be **HEALTHY**.
7. Sign in to the Chef Automate dashboard with your existing credentials, and verify that your nodes are reporting correctly.

Upgrade to Chef Automate 2 (CLI)

1. (Optional) If you aren't sure which of your AWS OpsWorks for Chef Automate servers are eligible for upgrade, run the following command. Be sure to add the `--region` parameter if you want to list AWS OpsWorks for Chef Automate servers in an AWS Region that is different from your default AWS Region.

   ```
   aws opsworks-cm describe-servers
   ```

   In the results, look for the a value of `true` for the attribute `CHEF_MAJOR_UPGRADE_AVAILABLE`. This indicates that the server is eligible for upgrade to Chef Automate 2. Make a note of the names of AWS OpsWorks for Chef Automate servers that are eligible for upgrade.
2. Run the following command, replacing *`server_name`* with the name of an AWS OpsWorks for Chef Automate server. To upgrade to Chef Automate 2 instead of performing routine system maintenance, add the `CHEF_MAJOR_UPGRADE` engine attribute, as shown in the command. Add the `--region` parameter if the target server is not in your default AWS Region. You can only upgrade one server per command.

   ```
   aws opsworks-cm start-maintenance --server-name server_name --engine-attributes
    Name=CHEF_MAJOR_UPGRADE,Value=true --region region
   ```

   If AWS OpsWorks for Chef Automate cannot upgrade the server for any reason, this command results in a validation exception.
3. Allow up to three hours for the upgrade. You can check the upgrade status periodically by running the following command.

```
aws opsworks-cm describe-servers --server-name server_name
```

In the results, look for the `Status` value. A `Status` of `UNDER_MAINTENANCE` means that the upgrade is still in progress. A successful upgrade returns messages similar to the following.

```
2019/10/24 00:27:56 UTC          Successfully upgraded to Automate 2.
2019/10/23 23:50:38 UTC          Upgrading Chef server from Automate 1 to Automate
 2
```

If the upgrade was unsuccessful, AWS OpsWorks for Chef Automate automatically rolls back your server to Chef Automate 1.

If the upgrade was successful but the server is not functioning the same as before the upgrade (for example, if managed nodes are not reporting), you can roll the server back manually. For manual rollback information, see Roll Back an AWS OpsWorks for Chef Automate Server to Chef Automate 1 (CLI).

Roll Back an AWS OpsWorks for Chef Automate Server to Chef Automate 1 (CLI)

If the upgrade process fails, AWS OpsWorks for Chef Automate automatically rolls your server back to Chef Automate 1. If the upgrade was successful but the server is not functioning the same as before the upgrade, you can roll your AWS OpsWorks for Chef Automate server back to Chef Automate 1 manually by using the AWS CLI.

1. Run the following command to show the `BackupId` of the last backup performed on your server before you attempted the upgrade. Add the `--region` parameter if your server is in an AWS Region that is different from your default AWS Region.

   ```
   aws opsworks-cm describe-backups server_name
   ```

   Backup IDs are in the format *`ServerName-yyyyMMddHHmmssSSS`*. Look for the following Chef Automate 1 properties in the results.

   ```
   "Engine": "Chef"
   "EngineVersion": "12.17.33"
   ```

2. Run the following command, using the backup ID you returned in step 1 as the value of `--backup-id`.

   ```
   aws opsworks-cm restore-server --server-name server_name --backup-id ServerName-yyyyMMddHHmmssSSS
   ```

   Allow between 20 minutes and three hours to restore the server, depending on the amount of data you stored on the server. During the restore operation, your server has a status of `RESTORING`. This status is displayed on the server's properties page in the AWS Management Console, and returned in the results of the **describe-servers** command.

3. After restoration is finished, the console displays the message, **Restore completed successfully**. Your AWS OpsWorks for Chef Automate server is online, and the same as it was before you started the upgrade process.

See Also

- System Maintenance in AWS OpsWorks for Chef Automate
- Restore an AWS OpsWorks for Chef Automate Server from a Backup
- **DescribeServers** in the *OpsWorks API Reference*
- **StartMaintenance** in the *OpsWorks API Reference*

Getting Started with AWS OpsWorks for Chef Automate

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

AWS OpsWorks for Chef Automate lets you run a Chef Automate server in AWS. You can provision a Chef server in about 15 minutes.

Starting May 3, 2021, AWS OpsWorks for Chef Automate stores some Chef Automate server attributes in AWS Secrets Manager. For more information, see Integration with AWS Secrets Manager.

The following walkthrough helps you create your first Chef server in AWS OpsWorks for Chef Automate.

Prerequisites

Before you begin, you must complete the following prerequisites.

Topics

- Set Up a VPC
- Prerequisites for Using a Custom Domain (Optional)
- Set Up an EC2 Key Pair (Optional)

Set Up a VPC

Your AWS OpsWorks for Chef Automate server must operate in an Amazon Virtual Private Cloud. You can add it to an existing VPC, use the default VPC, or create a new VPC to contain the server. For information about Amazon VPC and how to create a new VPC, see the Amazon VPC Getting Started Guide.

If you create your own VPC, or use an existing one, the VPC should have the following settings or properties.

- The VPC should have at least one subnet.

  If your AWS OpsWorks for Chef Automate server will be publicly accessible, make the subnet public, and enable **Auto-assign public IP**.
- **DNS resolution** should be enabled.
- On the subnet, enable **Auto-assign public IP**.

Prerequisites for Using a Custom Domain (Optional)

You can set up your Chef Automate server on your own domain, specifying a public endpoint in a custom domain to use as the endpoint of your server. When you use a custom domain, all of the following are required, as described in detail in this section.

Topics

- Set Up a Custom Domain
- Get a Certificate
- Get a Private Key

Set Up a Custom Domain

To run your Chef Automate server on your own custom domain, you will need a public endpoint of a server, such as `https://aws.my-company.com`. If you specify a custom domain, you must also provide a certificate and a private key, as described in the preceding sections.

To access the server after you create it, add a CNAME DNS record in your preferred DNS service. This record must point the custom domain to the endpoint (the value of the server's `Endpoint` attribute) that is generated by the Chef Automate server creation process. You cannot access the server by using the generated `Endpoint` value if the server is using a custom domain.

Get a Certificate

To set up your Chef Automate server on your own custom domain, you need A PEM-formatted HTTPS certificate. This can be be a single, self-signed certificate, or a certificate chain. As you complete the **Create Chef Automate server** workflow, if you specify this certificate, you must also provide a custom domain and a private key.

The following are requirements for the certificate value:

- You can provide either a self-signed, custom certificate, or the full certificate chain.
- The certificate must be a valid X509 certificate, or a certificate chain in PEM format.
- The certificate must be valid at the time of upload. A certificate can't be used before its validity period begins (the certificate's `NotBefore` date), or after it expires (the certificate's `NotAfter` date).
- The certificate’s common name or subject alternative names (SANs), if present, must match the custom domain value.
- The certificate must match the value of the **Custom private key** field.

Get a Private Key

To set up your Chef Automate server on your own custom domain, you need a private key in PEM format for connecting to the server by using HTTPS. The private key must not be encrypted; it cannot be protected by a password or passphrase. If you specify a custom private key, you must also provide a custom domain and a certificate.

Set Up an EC2 Key Pair (Optional)

An SSH connection is not necessary or recommended for typical management of the Chef server; you can use `knife` commands to perform most management tasks on your Chef server.

An EC2 key pair is required to connect to your server by using SSH in the event that you lose or want to change the sign-in password for the Chef Automate dashboard. You can use an existing key pair, or create a new key pair. For more information about how to create a new EC2 key pair, see Amazon EC2 Key Pairs.

If you don't need an EC2 key pair, you are ready to create a Chef server.

Create a Chef Automate Server

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

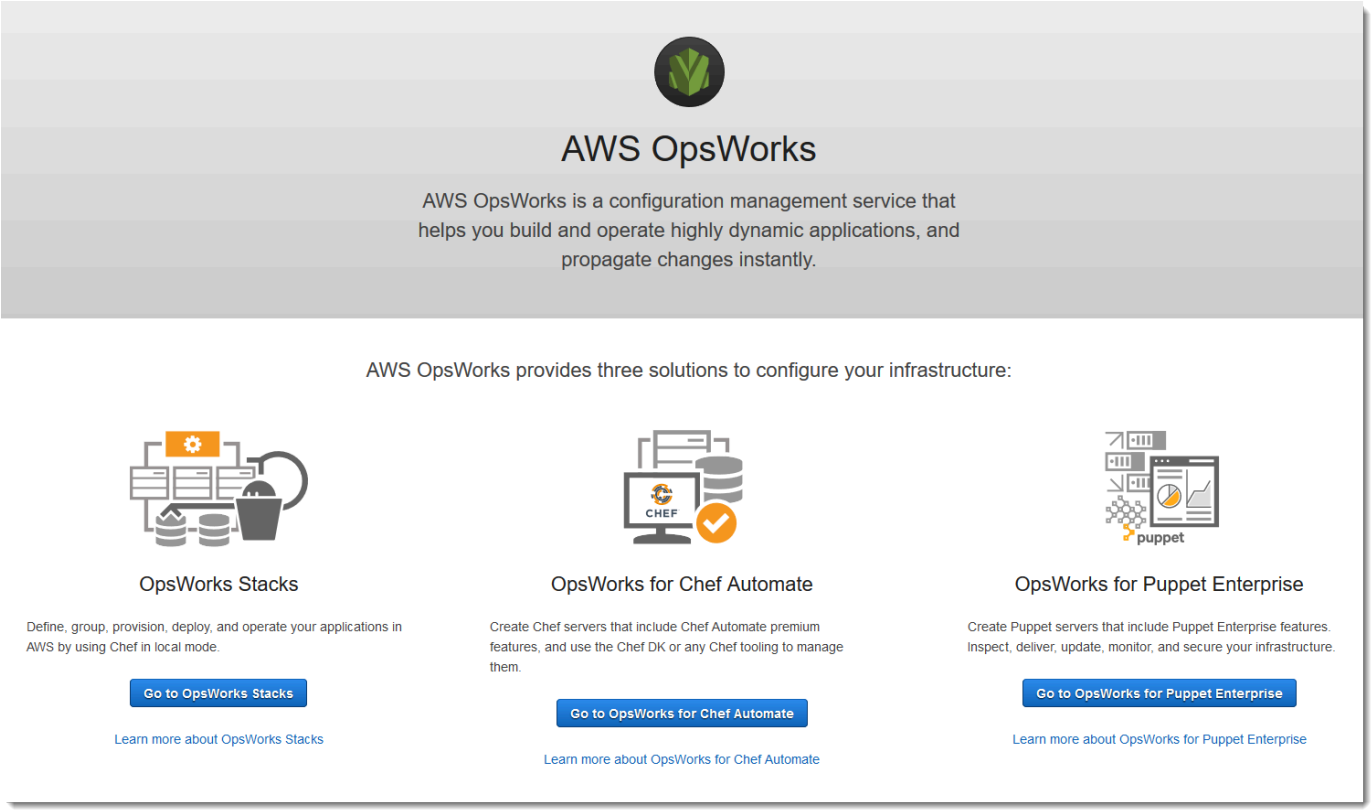
You can create a Chef server by using the AWS OpsWorks for Chef Automate console, or the AWS CLI.

Topics

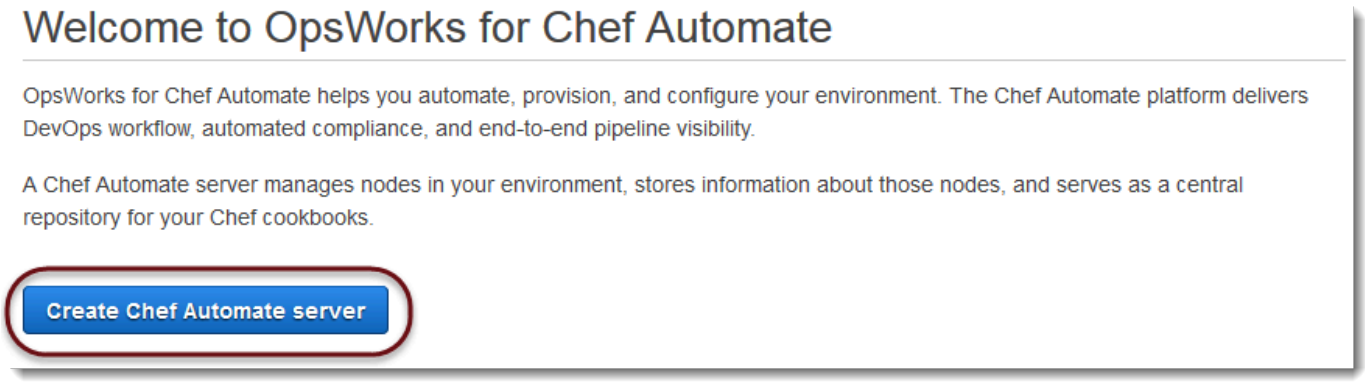
- Create a Chef Automate server in the AWS Management Console
- Create a Chef Automate server by using the AWS CLI

Create a Chef Automate server in the AWS Management Console

1. Sign in to the AWS Management Console and open the OpsWorks console at https://console.aws.amazon.com/opsworks/.
2. On the OpsWorks home page, choose **Go to OpsWorks for Chef Automate**.

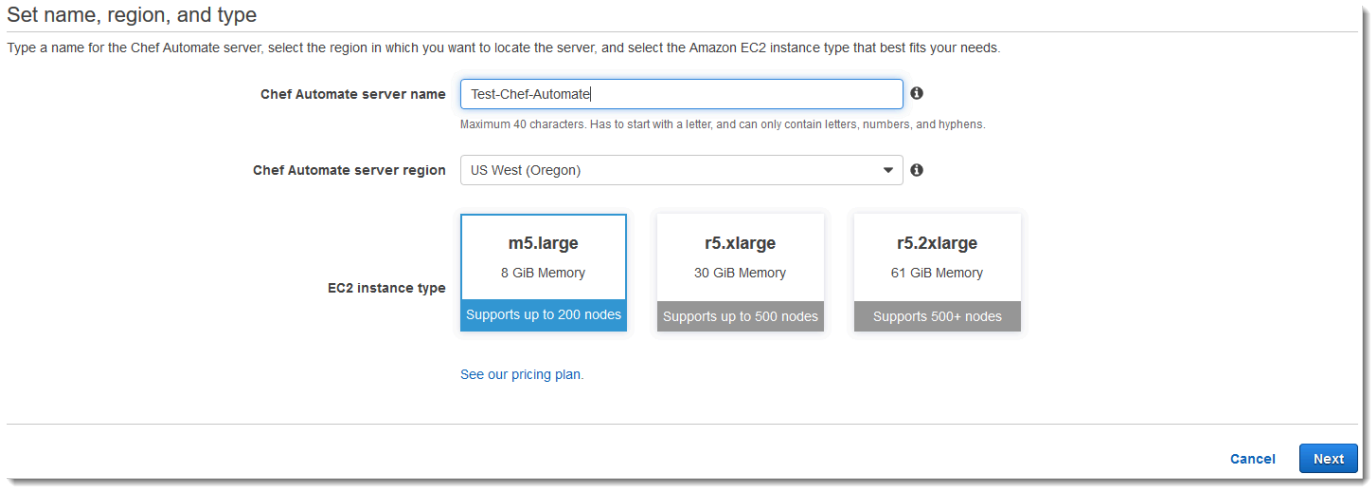


3. On the AWS OpsWorks for Chef Automate home page, choose **Create Chef Automate server**.

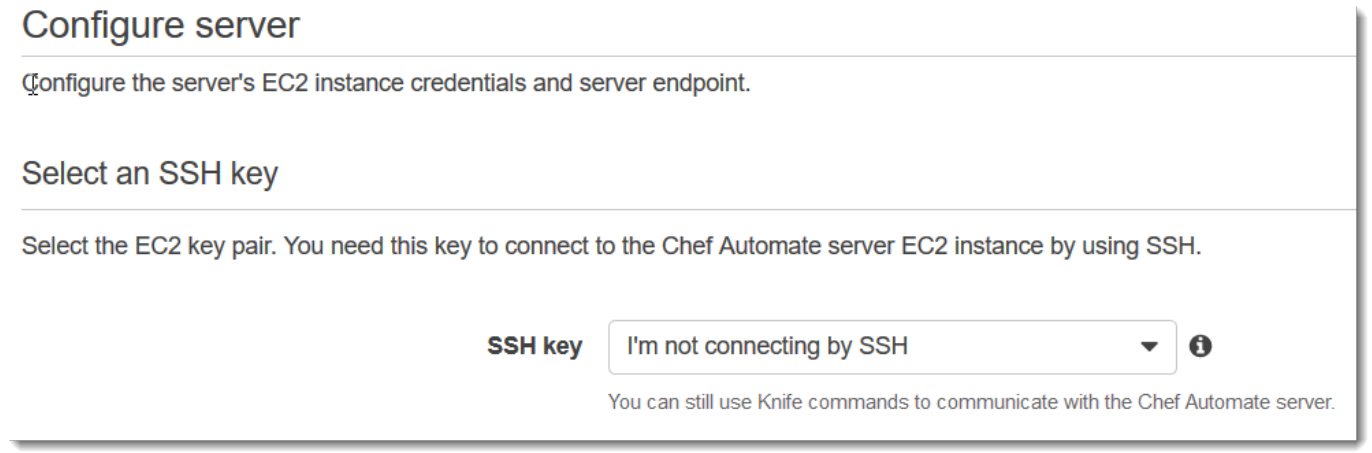


4. On the **Set name, region, and type** page, specify a name for your server. Chef server names can be a maximum of 40 characters, and can contain only alphanumeric characters and dashes. Select a supported region, and then choose an instance type that supports the number of nodes that you want to manage. You can change the instance type after your server has been

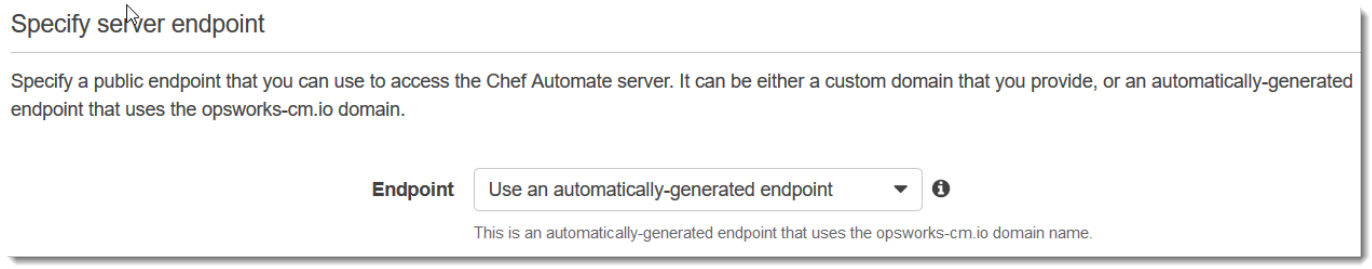
created, if needed. For this walkthrough, we are creating an **m5.large** instance type in the US West (Oregon) Region. Choose **Next**.



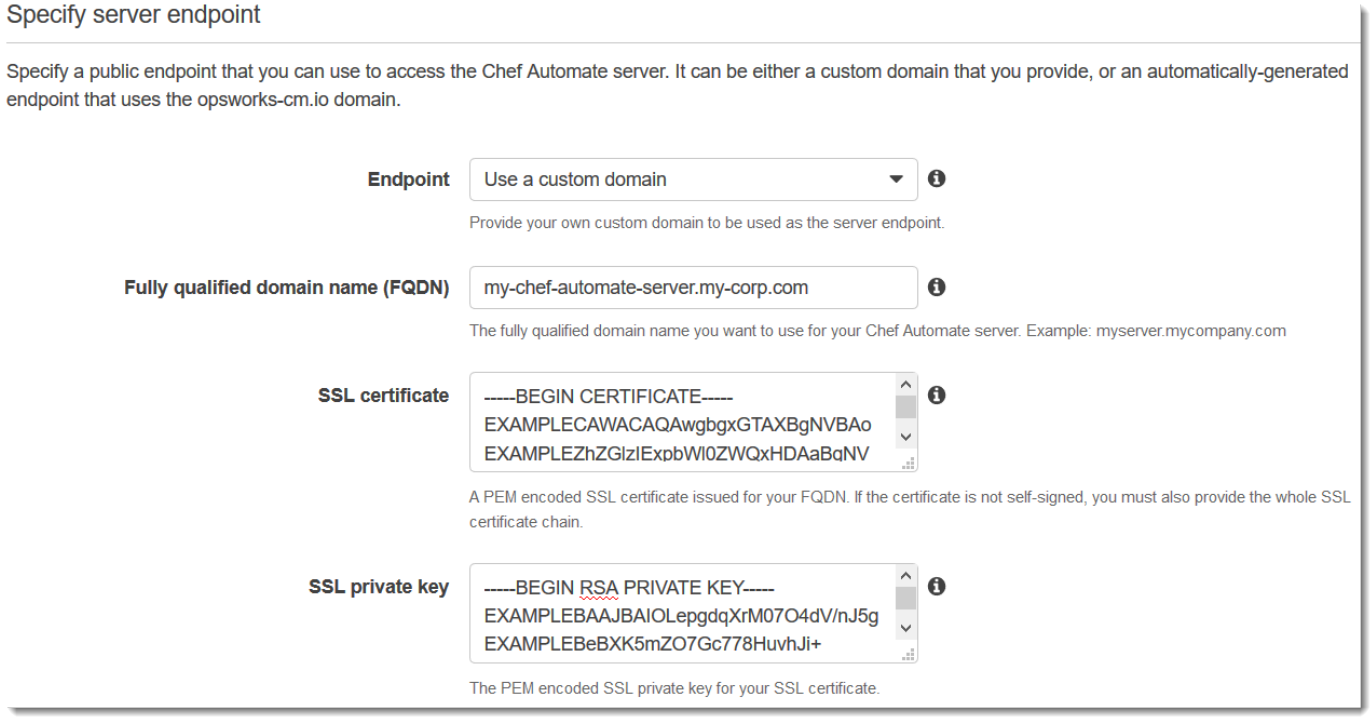
5. On the **Configure server** page, leave the default selection in the **SSH key** drop-down list, unless you want to specify a key pair name.



6. For **Specify server endpoint**, leave the default, **Use an automatically-generated endpoint** and then choose **Next**, unless you want your server on a custom domain of your own. To configure a custom domain, go on to the next step.



7. To use a custom domain, for **Specify server endpoint**, choose **Use a custom domain** from the drop-down list.



a. For **Fully qualified domain name (FQDN)**, specify an FQDN. You must own the domain name that you want to use.

b. For **SSL certificate**, paste in the entire PEM-formatted certificate, beginning with `-----BEGIN CERTIFICATE-----` and ending with `-----END CERTIFICATE-----`. The SSL certificate subject must match the FQDN you entered in the preceding step.

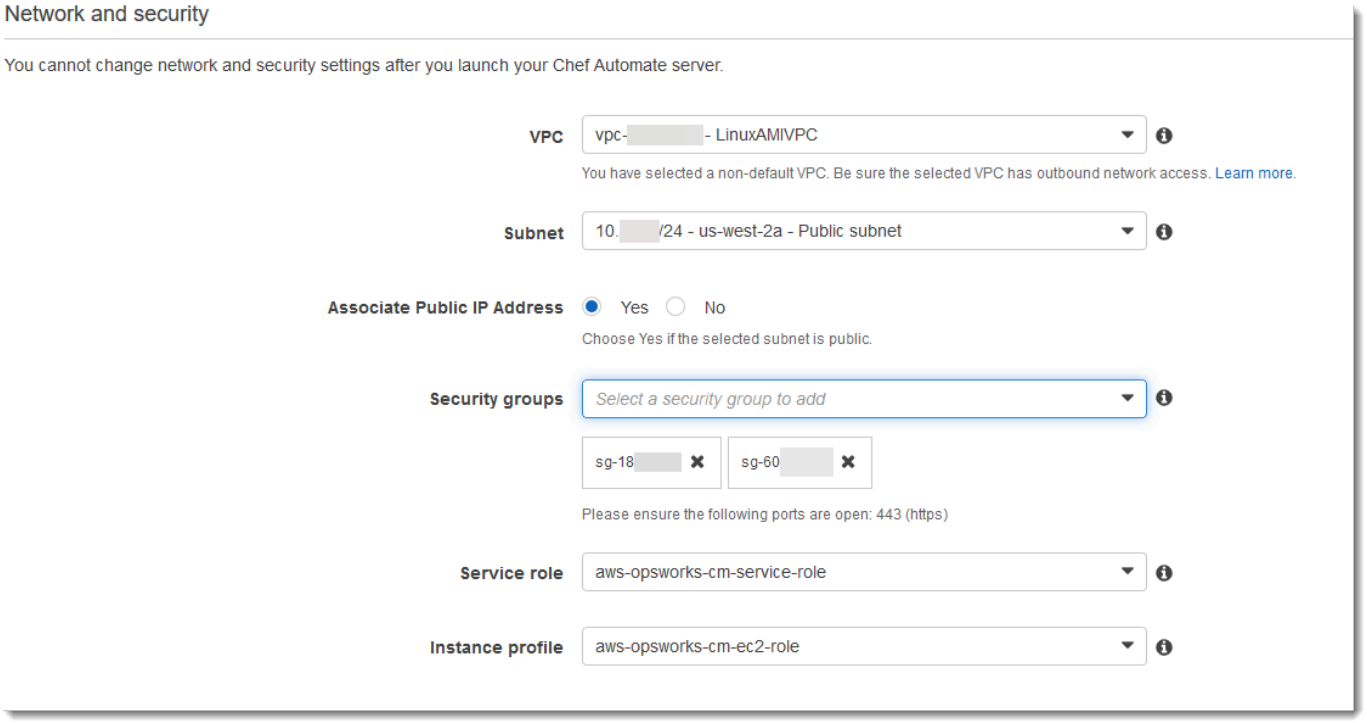
c. For **SSL private key**, paste in the entire RSA private key, beginning with `-----BEGIN RSA PRIVATE KEY-----` and ending with `-----END RSA PRIVATE KEY-----`. The SSL private key must match the public key in the SSL certificate that you entered in the preceding step. Choose **Next**.

8. On the **Configure Advanced Settings** page, in the **Network and Security** area, choose a VPC, subnet, and one or more security groups. The following are requirements for your VPC:

   - The VPC must have at least one public subnet.
   - DNS resolution must be enabled.
   - **Auto-assign public IP** must be enabled on public subnets.

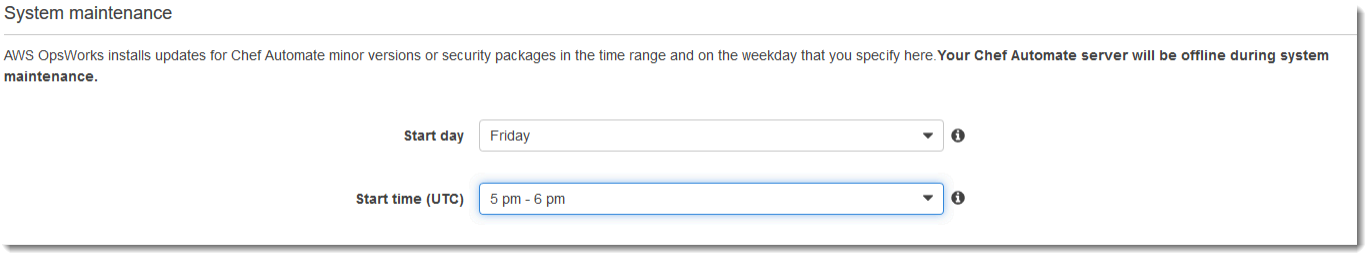
   OpsWorks can generate a security group, service role, and instance profile for you, if you do not already have ones that you want to use. Your server can be a member of multiple security

groups. You cannot change network and security settings for the Chef server after you have left this page.

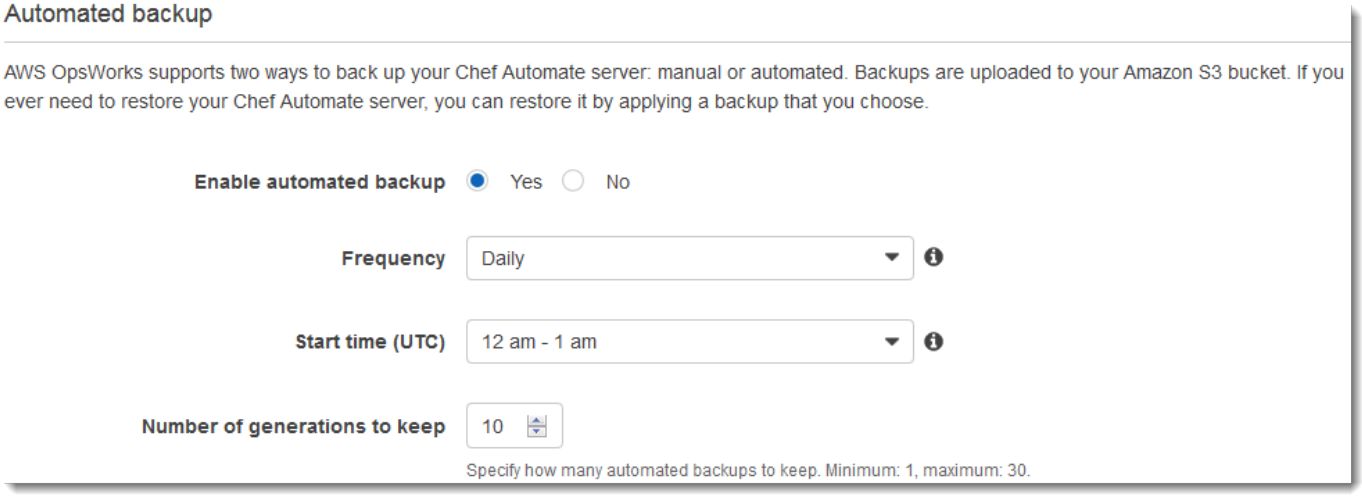


9. In the **System maintenance** section, set the day and hour that you want system maintenance to begin. Because you should expect the server to be offline during system maintenance, choose a time of low server demand within regular office hours. Connected nodes enter a `pending-server` state until maintenance is complete.

   The maintenance window is required. You can change the start day and time later by using the AWS Management Console, AWS CLI, or the APIs.



10. Configure backups. By default, automatic backups are enabled. Set a preferred frequency and hour for automatic backup to start, and set the number of backup generations to store in Amazon Simple Storage Service. A maximum of 30 backups are kept; when the maximum is reached, AWS OpsWorks for Chef Automate deletes the oldest backups to make room for new ones.



11. (Optional) In **Tags**, add tags to the server and related resources, such as the EC2 instance, Elastic IP address, security group, S3 bucket, and backups. For more information about tagging an AWS OpsWorks for Chef Automate server, see Working with Tags on AWS OpsWorks for Chef Automate Resources.
12. When you are finished configuring advanced settings, choose **Next**.
13. On the **Review** page, review your choices. When you are ready to create the server, choose **Launch**.

    While you are waiting for OpsWorks to create your Chef server, go on to Configure the Chef Server Using the Starter Kit and download the Starter Kit and the Chef Automate dashboard credentials. Do not wait until your server is online to download these items.

    When server creation is finished, your Chef server is available on the AWS OpsWorks for Chef Automate home page, with a status of **online**. After the server is online, the Chef Automate dashboard is available on the server's domain, at a URL in the following format: `https://`*`your_server_name`*`-random.region.opsworks-cm.io`.

Create a Chef Automate server by using the AWS CLI

Creating an AWS OpsWorks for Chef Automate server by running AWS CLI commands differs from creating a server in the console. In the console, OpsWorks creates a service role and security group for you, if you do not specify existing ones that you want to use. In the AWS CLI, OpsWorks can create a security group for you if you do not specify one, but it does not automatically create a service role; you must provide a service role ARN as part of your `create-server` command. In the console, while OpsWorks is creating your Chef Automate server, you download the Chef Automate

starter kit and the sign-in credentials for the Chef Automate dashboard. Because you cannot do this when you create an AWS OpsWorks for Chef Automate server by using the AWS CLI, you use a JSON processing utility to get the sign-in credentials and the starter kit from the results of the `create-server` command after your new AWS OpsWorks for Chef Automate server is online. Alternatively, you can generate a new set of sign-in credentials and a new starter kit in the console after your new AWS OpsWorks for Chef Automate server is online.

If your local computer is not already running the AWS CLI, download and install the AWS CLI by following installation instructions in the *AWS Command Line Interface User Guide*. This section does not describe all parameters that you can use with the `create-server` command. For more information about `create-server` parameters, see `create-server` in the *AWS CLI Reference*.

1. Be sure to complete the prerequisites, especially Set Up a VPC, or be sure that you have an existing VPC that you want to use. To create your Chef Automate server, you need a subnet ID.
2. Optionally, generate a Chef pivotal key by using OpenSSL, and save the key to a secure, convenient file on your local computer. The pivotal key is automatically generated as part of the server creation process if you do not provide one in the `create-server` command. If you want to skip this step, you can instead get the Chef Automate pivotal key from the results of the `create-server` command. If you choose to generate the pivotal key using the following commands, be sure to include the `-pubout` parameter, because the Chef Automate pivotal key value is the public half of the RSA key pair. For more information, see Step 6.

   ```
   umask 077
   openssl genrsa -out "pivotal" 2048
   openssl rsa -in "pivotal" -pubout
   ```
3. Create a service role and an instance profile.
4. Find and copy the ARNs of service roles in your account.

   ```
   aws iam list-roles --path-prefix "/service-role/" --no-paginate
   ```

   In the results of the `list-roles` command, look for service role ARN entries that resemble the following. Make a note of the service role ARNs. You need these values to create your Chef Automate server.
5. Find and copy the ARNs of instance profiles in your account.

   ```
   aws iam list-instance-profiles --no-paginate
   ```

In the results of the `list-instance-profiles` command, look for instance profile ARN entries that resemble the following. Make a note of the instance profile ARNs. You need these values to create your Chef Automate server.

6. Create the AWS OpsWorks for Chef Automate server by running the `create-server` command.

   - The `--engine` value is `ChefAutomate`, `--engine-model` is `Single`, and `--engine-version` is 12.
   - The server name must be unique within your AWS account, within each region. Server names must start with a letter; then letters, numbers, or hyphens (-) are allowed, up to a maximum of 40 characters.
   - Use the instance profile ARN and service role ARN that you copied in Steps 4 and 5.
   - Valid instance types are `m5.large`, `r5.xlarge`, or `r5.2xlarge`. For more information about the specifications of these instance types, see Instance Types in the *Amazon EC2 User Guide*.
   - The `--engine-attributes` parameter is optional; if you don't specify one or both values, the server creation process generates the values for you. If you add `--engine-attributes`, specify either the `CHEF_AUTOMATE_PIVOTAL_KEY` value that you generated in Step 2, a `CHEF_AUTOMATE_ADMIN_PASSWORD`, or both.

     If you do not set a value for `CHEF_AUTOMATE_ADMIN_PASSWORD`, a password is generated and returned as part of the `create-server` response. You can also download the starter kit again in the console, which regenerates this password. The password length is a minimum of eight characters, and a maximum of 32. The password can contain letters, numbers, and special characters (`!/@#$%^+=_`). The password must contain at least one lower case letter, one upper case letter, one number, and one special character.
   - An SSH key pair is optional, but can help you connect to your Chef Automate server if you need to reset the Chef Automate dashboard administrator password. For more information about creating an SSH key pair, see Amazon EC2 Key Pairs in the *Amazon EC2 User Guide*.
   - To use a custom domain, add the following parameters to your command. Otherwise, the Chef Automate server creation process automatically generates an endpoint for you. All three parameters are required to configure a custom domain. For information about additional requirements for using these parameters, see CreateServer in the OpsWorks CM API Reference.

  - `--custom-domain` - An optional public endpoint of a server, such as `https://aws.my-company.com`.
  - `--custom-certificate` - A PEM-formatted HTTPS certificate. The value can be be a single, self-signed certificate, or a certificate chain.
  - `--custom-private-key` - A private key in PEM format for connecting to the server by using HTTPS. The private key must not be encrypted; it cannot be protected by a password or passphrase.
- Weekly system maintenance is required. Valid values must be specified in the following format: `DDD:HH:MM`. The specified time is in coordinated universal time (UTC). If you do not specify a value for `--preferred-maintenance-window`, the default value is a random, one-hour period on Tuesday, Wednesday, or Friday.
- Valid values for `--preferred-backup-window` must be specified in one of the following formats: `HH:MM` for daily backups, or `DDD:HH:MM` for weekly backups. The specified time is in UTC. The default value is a random, daily start time. To opt out of automatic backups, add the parameter `--disable-automated-backup` instead.
- For `--security-group-ids`, enter one or more security group IDs, separated by a space.
- For `--subnet-ids`, enter a subnet ID.

```
aws opsworks-cm create-server --engine "ChefAutomate" --engine-model "Single"
 --engine-version "12" --server-name "server_name" --instance-profile-arn
 "instance_profile_ARN" --instance-type "instance_type" --engine-attributes
 '{"CHEF_AUTOMATE_PIVOTAL_KEY":"pivotal_key","CHEF_AUTOMATE_ADMIN_PASSWORD":"password"}'
 --key-pair "key_pair_name" --preferred-maintenance-window
 "ddd:hh:mm" --preferred-backup-window "ddd:hh:mm" --security-group-
ids security_group_id1 security_group_id2 --service-role-arn "service_role_ARN" --
subnet-ids subnet_ID
```

The following is an example.

```
aws opsworks-cm create-server --engine "ChefAutomate" --engine-
model "Single" --engine-version "12" --server-name "automate-06" --
instance-profile-arn "arn:aws:iam::12345678912:instance-profile/aws-
opsworks-cm-ec2-role" --instance-type "m5.large" --engine-attributes
 '{"CHEF_AUTOMATE_PIVOTAL_KEY":"MZZE...Wobg","CHEF_AUTOMATE_ADMIN_PASSWORD":"zZZzDj2DLYXSZF
 --key-pair "amazon-test" --preferred-maintenance-window "Mon:08:00" --preferred-
backup-window "Sun:02:00" --security-group-ids sg-b00000001 sg-b0000008 --service-
```

```
role-arn "arn:aws:iam::12345678912:role/service-role/aws-opsworks-cm-service-role"
 --subnet-ids subnet-300aaa00
```

The following example creates a Chef Automate server that uses a custom domain.

```
aws opsworks-cm create-server --engine "ChefAutomate" --engine-model "Single" --
engine-version "12" \
    --server-name "my-custom-domain-server" \
    --instance-profile-arn "arn:aws:iam::12345678912:instance-profile/aws-opsworks-
cm-ec2-role" \
    --instance-type "m5.large" \
    --engine-attributes
 '{"CHEF_AUTOMATE_PIVOTAL_KEY":"MZZE...Wobg","CHEF_AUTOMATE_ADMIN_PASSWORD":"zZZzDj2DLYXSZF
 \
    --custom-domain "my-chef-automate-server.my-corp.com" \
    --custom-certificate "-----BEGIN CERTIFICATE----- EXAMPLEqEXAMPLE== -----END
 CERTIFICATE-----" \
    --custom-private-key "-----BEGIN RSA PRIVATE KEY----- EXAMPLEqEXAMPLE= -----END
 RSA PRIVATE KEY-----" \
    --key-pair "amazon-test" \
    --preferred-maintenance-window "Mon:08:00" \
    --preferred-backup-window "Sun:02:00" \
    --security-group-ids sg-b00000001 sg-b0000008 \
    --service-role-arn "arn:aws:iam::12345678912:role/service-role/aws-opsworks-cm-
service-role" \
    --subnet-ids subnet-300aaa00
```

The following example creates a Chef Automate server that adds two tags: `Stage: Production` and `Department: Marketing`. For more information about adding and managing tags on AWS OpsWorks for Chef Automate servers, see Working with Tags on AWS OpsWorks for Chef Automate Resources in this guide.

```
aws opsworks-cm create-server --engine "ChefAutomate" --engine-model "Single" --
engine-version "12" \
    --server-name "my-test-chef-server" \
    --instance-profile-arn "arn:aws:iam::12345678912:instance-profile/aws-opsworks-
cm-ec2-role" \
    --instance-type "m5.large" \
    --engine-attributes
 '{"CHEF_AUTOMATE_PIVOTAL_KEY":"MZZE...Wobg","CHEF_AUTOMATE_ADMIN_PASSWORD":"zZZzDj2DLYXSZF
 \
    --key-pair "amazon-test" \
```

```
    --preferred-maintenance-window "Mon:08:00" \
    --preferred-backup-window "Sun:02:00" \
    --security-group-ids sg-b00000001 sg-b0000008 \
    --service-role-arn "arn:aws:iam::12345678912:role/service-role/aws-opsworks-cm-
service-role" \
    --subnet-ids subnet-300aaa00 \
    --tags [{\"Key\":\"Stage\",\"Value\":\"Production\"},{\"Key\":\"Department\",
\"Value\":\"Marketing\"}]
```

7. AWS OpsWorks for Chef Automate takes about 15 minutes to create a new server. Do not dismiss the output of the `create-server` command or close your shell session, because the output can contain important information that is not shown again. To get passwords and the starter kit from the `create-server` results, go on to the next step.

   If you are using a custom domain with the server, in the output of the `create-server` command, copy the value of the `Endpoint` attribute. The following is an example.

   ```
   "Endpoint": "automate-07-exampleexample.opsworks-cm.us-east-1.amazonaws.com"
   ```

8. If you opted to have AWS OpsWorks for Chef Automate generate a key and password for you, you can extract them in usable formats from the `create-server` results by using a JSON processor such as jq. After you install jq, you can run the following commands to extract the pivotal key, Chef Automate dashboard administrator password, and starter kit. If you did not provide your own pivotal key and password in Step 4, be sure to save the extracted pivotal key and administrator password in convenient but secure locations.

   ```
   #Get the Chef password:
   cat resp.json | jq -r '.Server.EngineAttributes[] | select(.Name ==
    "CHEF_AUTOMATE_ADMIN_PASSWORD") | .Value'

   #Get the Chef Pivotal Key:
   cat resp.json | jq -r '.Server.EngineAttributes[] | select(.Name ==
    "CHEF_AUTOMATE_PIVOTAL_KEY") | .Value'

   #Get the Chef Starter Kit:
   cat resp.json | jq -r '.Server.EngineAttributes[] | select(.Name ==
    "CHEF_STARTER_KIT") | .Value' | base64 -D > starterkit.zip
   ```

9. Optionally, if you did not extract the starter kit from `create-server` command results, you can download a new starter kit from the server's Properties page in the AWS OpsWorks for

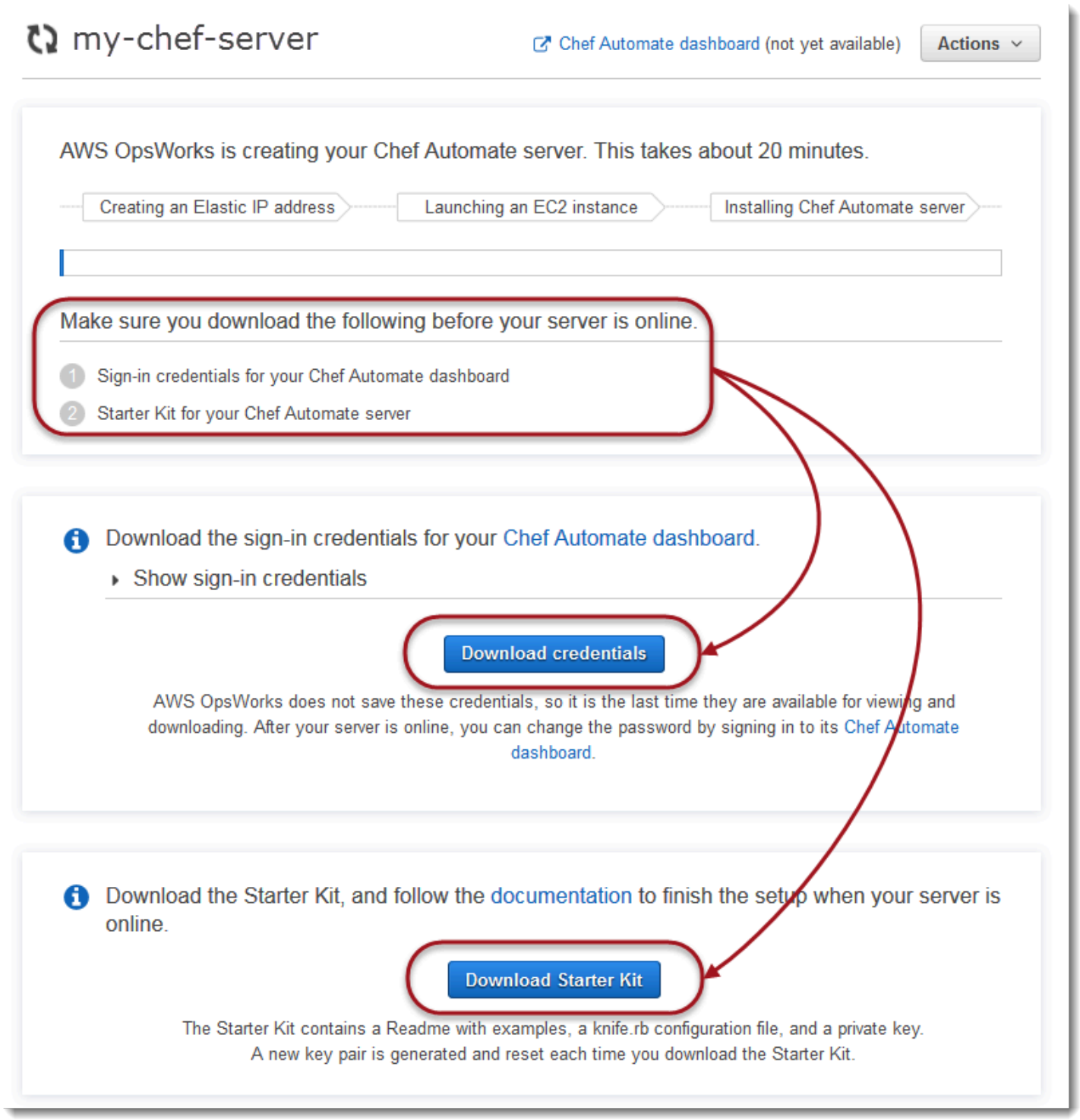
Chef Automate console. Downloading a new starter kit resets the Chef Automate dashboard administrator password.

10. If you are not using a custom domain, go on to the next step. If you are using a custom domain with the server, create a CNAME entry in your enterprise's DNS management tool to point your custom domain to the AWS OpsWorks for Chef Automate endpoint that you copied in step 7. You cannot reach or sign in to a server with a custom domain until you complete this step.
11. When the server creation process is finished, go on to the section called “Finish configuration and upload cookbooks”.

Configure the Chef Server Using the Starter Kit

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

While Chef server creation is still in progress, open its Properties page in the AWS OpsWorks for Chef Automate console. The first time that you work with a new Chef server, the Properties page prompts you to download two required items. Download these items before your Chef server is online; the download buttons are not available after a new server is online.



- **Sign-in credentials for the Chef server.** You will use these credentials to sign in to the Chef Automate dashboard, where you work with Chef Automate premium features, such as workflow and compliance scans. OpsWorks does not save these credentials; this is the last time that they

are available for viewing and downloading. If necessary, you can change the password that is provided with these credentials after you sign in.

- **Starter Kit.** The Starter Kit contains a README file with examples, a `knife.rb` configuration file, and a private key for the primary, or pivotal, user. A new key pair is generated—and the old key is reset—each time you download the Starter Kit.

In addition to the credentials that work only with the new server, the Starter Kit .zip file includes a simple example of a Chef repository that works with any AWS OpsWorks for Chef Automate server. In the Chef repository, you store cookbooks, roles, configuration files, and other artifacts for managing your nodes with Chef. We recommend that you store this repository in a version control system, such as Git, and treat it as source code. For information and examples that show how to set up a Chef repository that is tracked in Git, see About the chef-repo in the Chef documentation.

Prerequisites

1. While server creation is still in progress, download the sign-in credentials for the Chef server, and save them in a secure but convenient location.
2. Download the Starter Kit, and unzip the Starter Kit .zip file into your workspace directory. Do not share the Starter Kit private key. If other users will be managing the Chef server, add them as administrators in the Chef Automate dashboard later.
3. Download and install Chef Workstation (formerly known as the Chef Development Kit, or Chef DK) on the computer you will use to manage your Chef server and nodes. The `knife` utility is part of Chef Workstation. For instructions, see Install Chef Workstation on the Chef website.

Explore the Starter Kit Contents

The Starter Kit has the following contents.

- `cookbooks/` - A directory for cookbooks that you create. The `cookbooks/` folder contains the `opsworks-webserver` cookbook, a wrapper cookbook that depends on the `nginx` cookbook from the Chef Supermarket website. `Policyfile.rb` defaults to Chef supermarket as a secondary source if cookbook dependencies are not available in the `cookbooks/` directory.
- `Policyfile.rb` - A Ruby-based policy file that defines the cookbooks, dependencies, and attributes that become the policy for your nodes.

- `userdata.sh` and `userdata.ps1` - You can use user data files to associate nodes automatically after launching your Chef Automate server. `userdata.sh` is for bootstrapping Linux-based nodes, and `userdata.ps1` is for Windows-based nodes.
- `Berksfile` - You can use this file if you prefer to use Berkshelf and `berks` commands to upload cookbooks and their dependencies. In this walkthrough, we use `Policyfile.rb` and Chef commands to upload cookbooks, dependencies, and attributes.
- `README.md`, a Markdown-based file that describes how to use the Starter Kit to set up your Chef Automate server for the first time.
- `.chef` is a hidden directory that contains a knife configuration file (`knife.rb`) and a secret authentication key file (.pem).
  - `.chef/knife.rb` - A knife configuration file (`knife.rb`). The `knife.rb` file is configured so that Chef's `knife` tool operations run against the AWS OpsWorks for Chef Automate server.
  - `.chef/ca_certs/opsworks-cm-ca-2020-root.pem` - A certification authority (CA)-signed SSL private key that is provided by OpsWorks. This key allows the server to identify itself to the Chef Infra client agent on nodes that your server manages.

Set Up Your Chef Repository

A Chef repository contains several directories. Each directory in the Starter Kit contains a README file that describes the directory's purpose, and how to use it for managing your systems with Chef. There are two ways to get cookbooks installed on your Chef server: running `knife` commands, or running a Chef command to upload a policy file (`Policyfile.rb`) to your server that downloads and installs specified cookbooks. This walkthrough uses Chef commands and `Policyfile.rb` to install cookbooks on your server.

1. Create a directory on your local computer for storing cookbooks, such as `chef-repo`. After you add cookbooks, roles, and other files to this repository, we recommend that you upload or store it in a secure, versioned system, such as CodeCommit, Git, or Amazon S3.
2. In the `chef-repo` directory, create the following directories:
   - `cookbooks/` - Stores cookbooks.
   - `roles/` - Stores roles in `.rb` or `.json` formats.
   - `environments/` - Stores environments in `.rb` or `.json` formats.

Use Policyfile.rb to Get Cookbooks from a Remote Source

In this section, edit `Policyfile.rb` to specify cookbooks, then run a Chef command to upload the file to your server and install cookbooks.

1. View `Policyfile.rb` in your Starter Kit. By default, `Policyfile.rb` includes the `opsworks-webserver` wrapper cookbook , which depends on the `nginx` cookbook available on the Chef Supermarket website. The `nginx` cookbook installs and configures a web server on managed nodes. The required `chef-client` cookbook, which installs the Chef Infra client agent on managed nodes, is also specified.

   `Policyfile.rb` also points to the optional Chef Audit cookbook, which you can use to set up compliance scans on nodes. For more information about setting up compliance scans and getting compliance results for managed nodes, see Compliance Scans in AWS OpsWorks for Chef Automate. If you do not want to configure compliance scans and auditing right now, delete `'audit'` from the `run_list` section, and do not specify the `audit` cookbook attributes at the end of the file.

```
# Policyfile.rb - Describe how you want Chef to build your system.
#
# For more information about the Policyfile feature, visit

# https://docs.chef.io/policyfile.html


# A name that describes what the system you're building with Chef does.

name 'opsworks-demo-webserver'

# The cookbooks directory is the preferred source for external cookbooks

default_source :chef_repo, "cookbooks/" do |s|

  s.preferred_for "nginx", "windows", "chef-client", "yum-epel", "seven_zip",
                  "build-essential", "mingw", "ohai", "audit", "logrotate", "cron"

end
# Alternative source
default_source :supermarket
```

```
# run_list: chef-client runs these recipes in the order specified.

run_list  'chef-client',
          'opsworks-webserver',
          'audit'
# add 'ssh-hardening' to your runlist to fix compliance issues detected by the ssh-
baseline profile

# Specify a custom source for a single cookbook:

cookbook 'opsworks-webserver', path: 'cookbooks/opsworks-webserver'


# Policyfile defined attributes

# Define audit cookbook attributes
default["opsworks-demo"]["audit"]["reporter"] = "chef-server-automate"
default["opsworks-demo"]["audit"]["profiles"] = [
  {
    "name": "DevSec SSH Baseline",
    "compliance": "admin/ssh-baseline"
  }
]
```

The following is an example of `Policyfile.rb` without the `audit` cookbook and attributes, if you want to configure only the `nginx` web server for now.

```
# Policyfile.rb - Describe how you want Chef to build your system.
#
# For more information on the Policyfile feature, visit
# https://docs.chef.io/policyfile.html

# A name that describes what the system you're building with Chef does.
name 'opsworks-demo-webserver'

# Where to find external cookbooks:
default_source :supermarket

# run_list: chef-client will run these recipes in the order specified.
run_list  'chef-client',
          'opsworks-webserver'

# Specify a custom source for a single cookbook:
```

```
cookbook 'opsworks-webserver', path: 'cookbooks/opsworks-webserver'
```

If you make changes to `Policyfile.rb`, be sure to save the file.

2. Download and install the cookbooks defined in `Policyfile.rb`.

```
chef install
```

All cookbooks are versioned in the cookbook's `metadata.rb` file. Each time you change a cookbook, you must raise the version of the cookbook that is in its `metadata.rb`.

3. If you have chosen to configure compliance scans, and kept the `audit` cookbook information in the policy file, push the policy `opsworks-demo` to your server.

```
chef push opsworks-demo
```

4. If you completed step 3, verify the installation of your policy. Run the following command.

```
chef show-policy
```

The results should resemble the following:

```
opsworks-demo-webserver
=======================
* opsworks-demo:  ec0fe46314
```

5. You are now ready to add or bootstrap nodes to your Chef Automate server. You can automate the association of nodes by following steps in Add nodes automatically in AWS OpsWorks for Chef Automate, or add nodes one at a time by following steps in Add nodes individually.

(Alternate) Use Berkshelf to Get Cookbooks from a Remote Source

Berkshelf is a tool for managing cookbooks and their dependencies. If you prefer to use Berkshelf instead of `Policyfile.rb` to install cookbooks into local storage, use the procedure in this section instead of the preceding section. You can specify which cookbooks and versions to use with your Chef server and upload them. The Starter Kit contains a file named `Berksfile` that you can use to list your cookbooks.

1. To get started, add the `chef-client` cookbook to the included Berksfile. The `chef-client` cookbook configures the Chef Infra client agent software on each node that you connect to

your Chef Automate server. To learn more about this cookbook, see Chef Client Cookbook in the Chef Supermarket.

2. Using a text editor, append another cookbook to your Berksfile that installs a web server application; for example, the `apache2` cookbook, which installs the Apache web server. Your Berksfile should resemble the following.

```
source 'https://supermarket.chef.io'
cookbook 'chef-client'
cookbook 'apache2'
```

3. Download and install the cookbooks on your local computer.

```
berks install
```

4. Upload the cookbook to the Chef server.

   On Linux, run the following.

```
SSL_CERT_FILE='.chef/ca_certs/opsworks-cm-ca-2020-root.pem' berks upload
```

   On Windows, run the following Chef Workstation command in a PowerShell session. Before you run the command, be sure to set the execution policy in PowerShell to `RemoteSigned`. Add `chef shell-init` to make Chef Workstation utility commands available to PowerShell.

```
$env:SSL_CERT_FILE="ca_certs\opsworks-cm-ca-2020-root.pem"
chef shell-init berks upload
Remove-Item Env:\SSL_CERT_FILE
```

5. Verify the installation of the cookbook by showing a list of cookbooks that are currently available on the Chef Automate server. You can do this by running the following `knife` command.

   You are ready to add nodes to manage with the AWS OpsWorks for Chef Automate server.

```
knife cookbook list
```

(Optional) Configure `knife` to Work with a Custom Domain

If your Chef Automate server uses a custom domain, you might need to add the PEM certificate of the root CA that signed your server's certificate chain, or your server PEM certificate if the certificate is self-signed. `ca_certs` is a subdirectory in `chef/` that contains certificate authorities (CAs) that are trusted by the Chef `knife` utility.

You can skip this section if you aren't using a custom domain, or if your custom certificate is signed by a root CA that is trusted by your operating system. Otherwise, configure `knife` to trust your Chef Automate server SSL certificate, as described in the following steps.

1. Run the following command.

   ```
   knife ssl check
   ```

   If the results are similar to the following, skip the rest of this procedure, and go on to Add Nodes for the Chef Server to Manage.

   ```
   Connecting to host my-chef-automate-server.my-corp.com:443
            Successfully verified certificates from 'my-chef-automate-server.my-
   corp.com'
   ```

   If you get an error message similar to the following, go on to the next step.

   ```
   Connecting to host my-chef-automate-server.my-corp.com:443
            ERROR: The SSL certificate of my-chef-automate-server.my-corp.com could
    not be verified.
            ...
   ```

2. Run `knife ssl fetch` to trust the certificates of your AWS OpsWorks for Chef Automate server. Alternatively, you can manually copy the root CA PEM-formatted certificate of your server to the directory that is the value of `trusted_certs_dir` in the output of `knife ssl check`. By default, this directory is in `.chef/ca_certs/` in the Starter Kit. Your output should resemble the following:

   ```
   WARNING: Certificates from my-chef-automate-server.my-corp.com will be fetched and
    placed in your trusted_cert
            directory (/Users/username/starterkit/.chef/../.chef/ca_certs).
   ```

```
          Knife has no means to verify these are the correct certificates. You
 should
          verify the authenticity of these certificates after downloading.

          Adding certificate for my-chef-automate-server in /Users/users/
starterkit/.chef/../.chef/ca_certs/servv-aqtswxu20swzkjgz.crt
          Adding certificate for MyCorp_Root_CA in /Users/users/
starterkit/.chef/../.chef/ca_certs/MyCorp_Root_CA.crt
```

3. Run `knife ssl check` again. Your output should resemble the following:

```
Connecting to host my-chef-automate-server.my-corp.com:443
          Successfully verified certificates from 'my-chef-automate-server.my-
corp.com'
```

You are ready to use `knife` with your Chef Automate server.

Add Nodes for the Chef Server to Manage

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The `chef-client` agent runs Chef recipes on physical or virtual computers, called *nodes*, that are associated with the server. You can connect on-premises computers or instances to the Chef server to manage, provided the nodes are running supported operating systems. Registering nodes with the Chef server installs the `chef-client` agent software on those nodes.

You can use the following methods to add nodes:

- Add notes individually by running a `knife` command that adds, or *bootstraps*, an EC2 instance so that the Chef server can manage it. For more information see Add nodes individually.
- Add nodes automatically by using a script to perform unattended association of nodes with the Chef server. The code in the Starter Kit shows how to add nodes automatically using the

unattended method. For more information see, Add nodes automatically in AWS OpsWorks for Chef Automate.

Topics

- Add nodes individually
- Add nodes automatically in AWS OpsWorks for Chef Automate

Add nodes individually

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section describes how to run a `knife` command that adds, or *bootstraps*, an EC2 instance so that the Chef server can manage it.

The minimum supported version of `chef-client` on nodes associated with an AWS OpsWorks for Chef Automate server is 13.*x*. We recommend running the most current, stable `chef-client` version.

Topics

- (Optional) Specify the URL of your Chef Automate Server Root CA
- Supported Operating Systems
- Add Nodes with Knife

(Optional) Specify the URL of your Chef Automate Server Root CA

If your server is using a custom domain and certificate, you might need to edit the `ROOT_CA_URL` variable in the userdata script with a public URL that you can use to get the root CA PEM-formatted certificate of your server. The following AWS CLI commands upload your root CA to an Amazon S3 bucket, and generate a presigned URL that you can use for one hour.

1. Upload the root CA PEM-formatted certificate to S3.

```
aws s3 cp ROOT_CA_PEM_FILE_PATH s3://bucket_name/
```

2. Generate a presigned URL that you can use for one hour (3600 seconds, in this example) to download the root CA.

```
aws s3 presign s3://bucket_name/ROOT_CA_PEM_FILE_NAME --expires-in 3600
```

3. Edit the variable ROOT_CA_URL in the userdata script with the value of the pre-signed URL.

Supported Operating Systems

For the current list of supported operating systems for nodes, see the Chef website.

Add Nodes with Knife

The `knife-ec2` plug-in is included with Chef Workstation. If you are more familiar with `knife-ec2`, you can use it instead of `knife bootstrap` to provision and bootstrap new EC2instances. Otherwise, launch a new EC2 instance, and then follow the steps in this section.

To add nodes to manage

1. Run the following `knife bootstrap` command. This command bootstraps an EC2 instance to the nodes that your Chef server will manage. Note that you are instructing the Chef server to run recipes from the `nginx` cookbook that you installed in the section called "Use Policyfile.rb to Get Cookbooks from a Remote Source". For more information about adding nodes by running the `knife bootstrap` command, see Bootstrap a Node in the Chef documentation.

   The following table shows valid user names for node operating systems in the `knife` command in this step. If neither `root` nor `ec2-user` works, check with your AMI provider. For more information about connecting to Linux-based instances, see Connecting to Your Linux Instance Using SSH in the AWS documentation.

   Valid values for user names in node operating systems

Operating System	Valid User Names
Amazon Linux	`ec2-user`

Operating System	Valid User Names
Red Hat Enterprise Linux 5	`root` or `ec2-user`
Ubuntu	`ubuntu`
Fedora	`fedora` or `ec2-user`
SUSE Linux	`root` or `ec2-user`

```
knife bootstrap INSTANCE_IP_ADDRESS -N INSTANCE_NAME -x USER_NAME --sudo --run-list
 "recipe[nginx]"
```

2. Verify that the new node was added by running the following commands, replacing *INSTANCE_NAME* with the name of the instance that you just added.

```
knife client show INSTANCE_NAME
knife node show INSTANCE_NAME
```

Add nodes automatically in AWS OpsWorks for Chef Automate

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This topic describes how to add Amazon Elastic Compute Cloud (Amazon EC2) nodes to your Chef server automatically. The code in the Starter Kit shows how to add nodes automatically using the unattended method. The recommended method of unattended (or automatic) association of new nodes is to configure the Chef Client Cookbook. You can use the `userdata` script in the Starter Kit, and change either the `run_list` section of the `userdata` script, or your `Policyfile.rb` with the cookbooks you want to apply to your nodes. Before you run the `chef-client` agent, install the Chef Client cookbook on your Chef server, and then install the `chef-client` agent in service mode with, for example, an HTTPD role, as shown in the following sample command.

```
chef-client -r "chef-client,role[httpd]"
```

To communicate with the Chef server, the `chef-client` agent software must have access to the public key of the client node. You can generate a public-private key pair in Amazon EC2, and then pass the public key to the OpsWorks `associate-node` API call with the node name. The script included in the Starter Kit gathers your organization name, server name, and server endpoint for you. This ensures that the node is associated with the Chef server, and the `chef-client` agent software that runs on the node can communicate with the server after matching the private key.

The minimum supported version of `chef-client` on nodes associated with an AWS OpsWorks for Chef Automate server is 13.*x*. We recommend running the most current, stable `chef-client` version.

For information about how to disassociate a node, see Disassociate a Node from an AWS OpsWorks for Chef Automate Server in this guide, and `disassociate-node` in the AWS OpsWorks for Chef Automate API documentation.

Topics

- Supported Operating Systems
- Step 1: Create an IAM Role to Use as Your Instance Profile
- Step 2: Install the Chef Client Cookbook
- Step 3: Create Instances by Using an Unattended Association Script
- Other Methods of Automating Repeated Runs of chef-client
- Related Topics

Supported Operating Systems

For the current list of supported operating systems for nodes, see the Chef website.

Step 1: Create an IAM Role to Use as Your Instance Profile

Create an AWS Identity and Access Management (IAM) role to use as your EC2 instance profile, and attach the following policy to the IAM role. This policy allows the AWS OpsWorks for Chef Automate (`opsworks-cm`) API to communicate with the EC2 instance during node registration. For more information about instance profiles, see Using Instance Profiles in the Amazon EC2 documentation. For information about how to create an IAM role, see Creating an IAM Role in the Console in the Amazon EC2 documentation.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "opsworks-cm:AssociateNode",
                "opsworks-cm:DescribeNodeAssociationStatus"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

Step 2: Install the Chef Client Cookbook

If you have not done so already, follow the steps in (Alternate) Use Berkshelf to Get Cookbooks from a Remote Source to ensure that your Berksfile or `Policyfile.rb` file references the Chef Client cookbook and installs the cookbook.

Step 3: Create Instances by Using an Unattended Association Script

1. To create EC2 instances, you can copy the `userdata` script from the Starter Kit to the `userdata` section of EC2 instance instructions, Amazon EC2 Auto Scaling group launch configurations, or an AWS CloudFormation template. For more information about adding scripts to user data, see Running Commands on Your Linux Instance at Launch in the Amazon EC2 documentation.

   This script runs the `opsworks-cm` API `associate-node` command to associate a new node with your Chef server.

   By default, the name of the new registered node is the instance ID, but you can change the name by modifying the value of the `NODE_NAME` variable in the `userdata` script. Because changing the organization name in the Chef console UI is currently not possible, leave `CHEF_AUTOMATE_ORGANIZATION` set to `default`.
2. Follow the procedure in Launching an Instance in the EC2 documentation, with modifications here. In the EC2 instance launch wizard, choose an Amazon Linux AMI.

3. On the **Configure Instance Details** page, select the role you created in Step 1: Create an IAM Role to Use as Your Instance Profile, as your IAM role.
4. In the **Advanced Details** area, upload the `userdata.sh` script that you created earlier in this procedure.
5. No changes are needed on the **Add Storage** page. Go on to **Add Tags**.
6. On the **Configure Security Group** page, choose **Add Rule**, and then choose the type **HTTP** to open port numbers 443 and 80 for the Apache web server in this example.
7. Choose **Review and Launch**, and then choose **Launch**. When your new node starts, it applies the configurations specified by the recipes you have specified in the `RUN_LIST` parameter.
8. Optional: If you have added the `nginx` cookbook to your run list, when you open the webpage linked to the public DNS of your new node, you should see a website that is hosted by your nginx web server.

Other Methods of Automating Repeated Runs of `chef-client`

Although more difficult to achieve, and not recommended, you can run the script in this topic solely as part of standalone instance user data, use a AWS CloudFormation template to add it to new instance user data, configure a `cron` job to run the script regularly, or run `chef-client` within a service. However, we recommend the Chef Client Cookbook method because there are some disadvantages with other automation techniques.

For a complete list of parameters you can provide to `chef-client`, see the Chef documentation.

Related Topics

The following AWS blog posts offer more information about automatically associating nodes with your Chef Automate server, by using Auto Scaling groups, or within multiple accounts.

- Using AWS OpsWorks for Chef Automate to Manage EC2 Instances with Auto Scaling
- OpsWorks for Chef Automate – Automatically Bootstrapping Nodes in Different Accounts

Sign in to the Chef Automate dashboard

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers

migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

After you have downloaded the sign-in credentials from the Chef server's Properties page, and the server is online, sign in to the Chef Automate dashboard. In this walkthrough, we instructed you to first upload cookbooks and add at least one node to manage. This allows you to see information about the cookbooks and nodes in the dashboard.

When you attempt to connect to the dashboard webpage, certificate warnings appear in your browser until you install an OpsWorks-specific, CA-signed SSL certificate on the client computer that you are using to manage your Chef server. If you prefer not to see the warnings before you continue to the dashboard webpage, install the SSL certificate before you sign in.

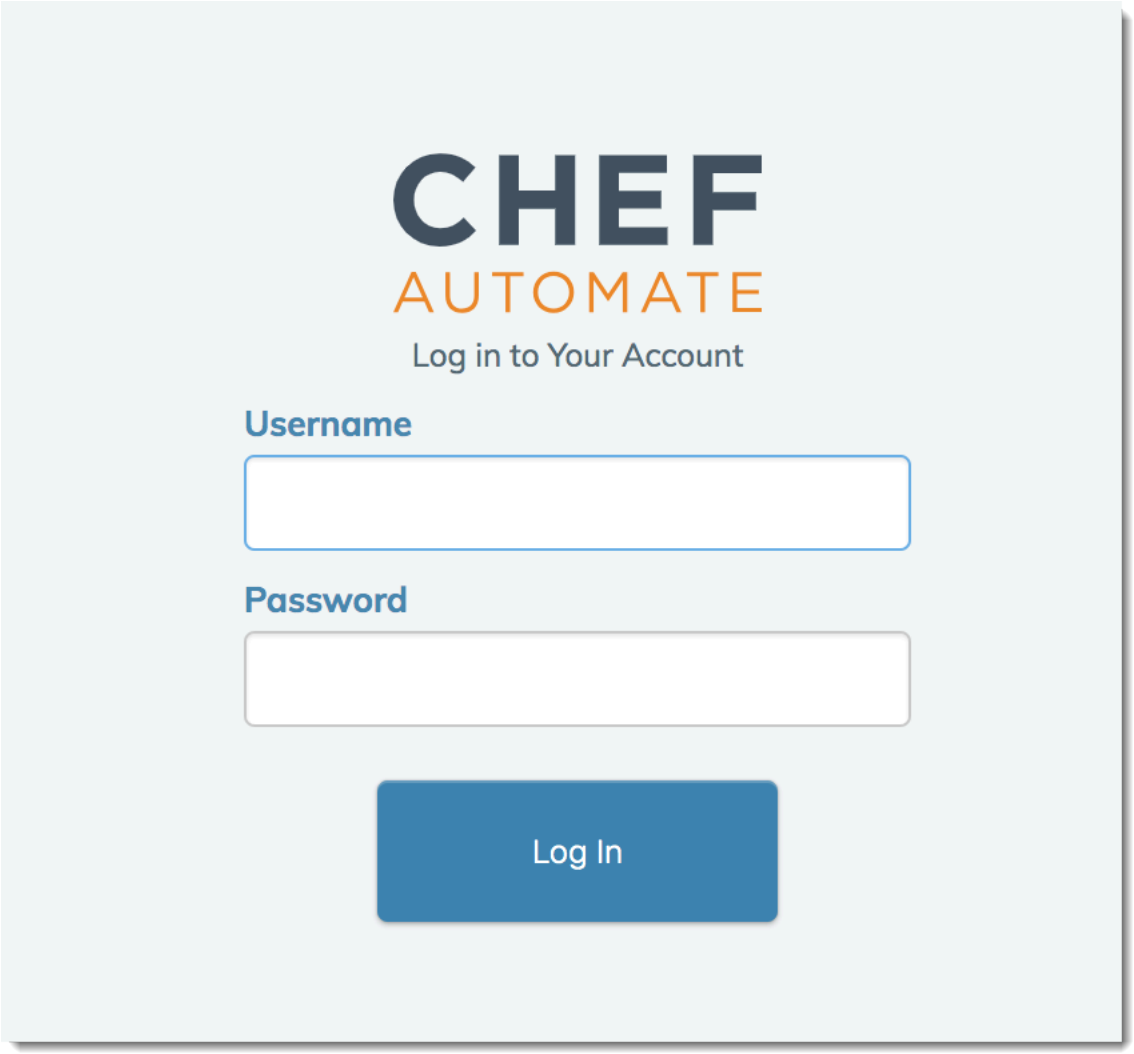
After you have installed the client-side SSL certificate, you can sign in to the Chef Automate dashboard without seeing warning messages.

Note

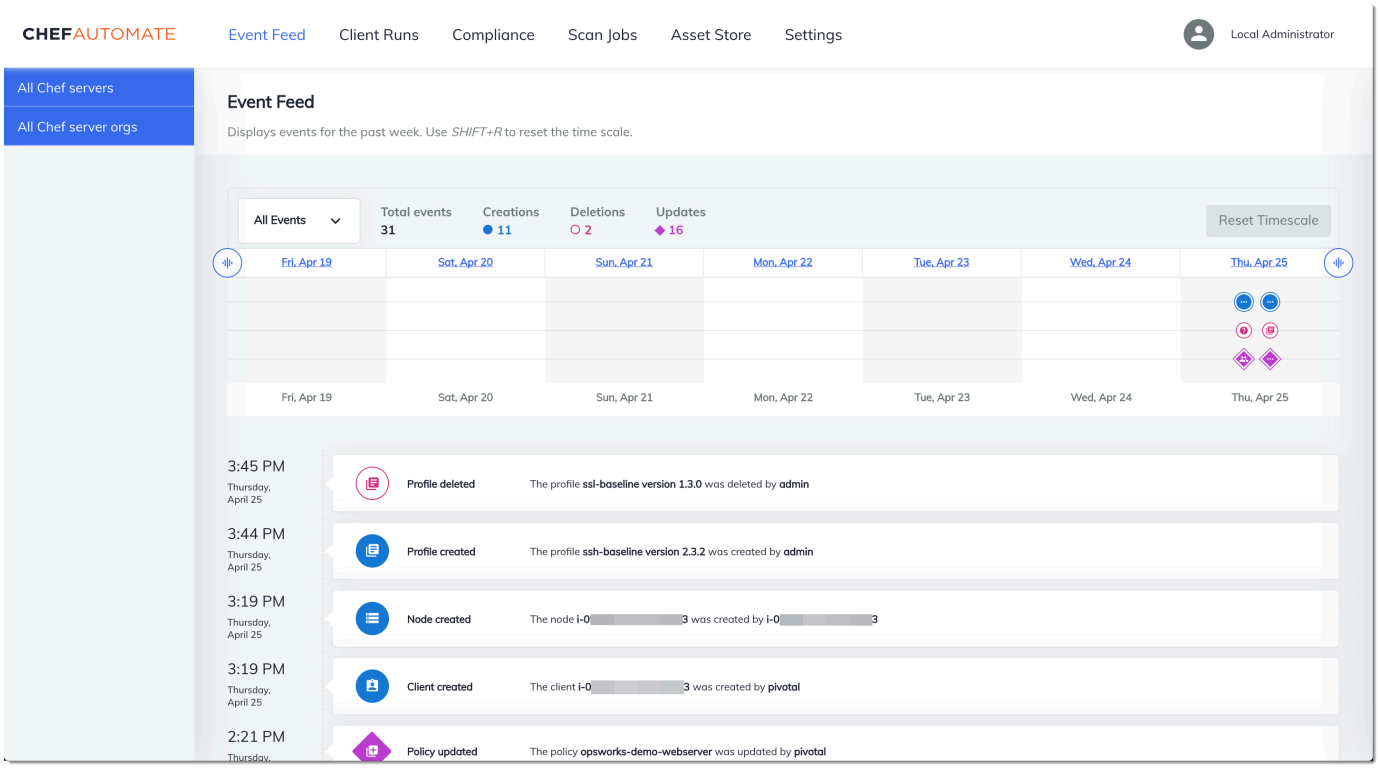
Users of Google Chrome on Ubuntu and Linux Mint operating systems can have difficulty signing in. We recommend that you use Mozilla Firefox or other browsers to sign in and use the Chef Automate dashboard on those operating systems. No issues have been found using Google Chrome on Windows or MacOS.

To sign in to the Chef Automate dashboard

1. Unzip and open the Chef Automate credentials that you downloaded in Prerequisites. You will need these credentials to sign in.
2. Open the **Properties** page for your Chef server.
3. At the upper right of the **Properties** page, choose **Open Chef Automate dashboard**.
4. Sign in using the credentials from Step 1.



5. In the Chef Automate dashboard, you can view detailed information about the nodes you've bootstrapped, cookbook run progress and events, the compliance level of nodes, and much more. For more information about the features of the Chef Automate dashboard and how to use them, see the Chef Automate Documentation.



Note

For information about how to change the password that you use to sign in to the Chef Automate dashboard, see Reset Chef Automate Dashboard Credentials.

Create an AWS OpsWorks for Chef Automate Server by using AWS CloudFormation

Important

AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

AWS OpsWorks for Chef Automate lets you run a Chef Automate server in AWS. You can provision a Chef Automate server in about 15 minutes.

Starting May 3, 2021, AWS OpsWorks for Chef Automate stores some Chef Automate server attributes in AWS Secrets Manager. For more information, see Integration with AWS Secrets Manager.

The following walkthrough helps you create a server in AWS OpsWorks for Chef Automate by creating a stack in AWS CloudFormation.

Topics

- Prerequisites
- Create a Chef Automate Server in AWS CloudFormation

Prerequisites

Before you create a new Chef Automate server, create the resources outside of AWS OpsWorks for Chef Automate that you'll need to access and manage your Chef server. For more information, see Prerequisites in the Getting Started section of this guide.

Review the OpsWorks-CM section of the *AWS CloudFormation User Guide* Template Reference to learn about the supported and required values in the AWS CloudFormation template that you use to create your server.

If you are creating a server that uses a custom domain, you need a custom domain, certificate, and private key. You must specify values for all three of these parameters in your AWS CloudFormation template. For more information about requirements for the `CustomDomain`, `CustomCertificate`, and `CustomPrivateKey` parameters, see CreateServer in the *OpsWorks CM API Reference*.

Create a password value for the `CHEF_AUTOMATE_ADMIN_PASSWORD` engine attribute. The password length is a minimum of eight characters, and a maximum of 32. The password can contain letters, numbers, and special characters (`!/@#$%^+=_`). The password must contain at least one lower case letter, one upper case letter, one number, and one special character. You specify this password in your AWS CloudFormation template, or as the value of the `CHEF_AUTOMATE_ADMIN_PASSWORD` parameter when you are creating your stack.

Generate a base64-encoded RSA key pair before you get started creating a Chef Automate server in AWS CloudFormation. The pair's public key is the value of `CHEF_AUTOMATE_PIVOTAL_KEY`,

the Chef- specific EngineAttributes from the CreateServer API. This key is provided as the value of **Parameters** in the AWS CloudFormation console, or in the **create-stack** command in the AWS CLI. To generate this key, we suggest the following methods.

- On Linux-based computers, you can generate this key by running the following OpenSSL command.

  ```
  openssl genrsa -out pivotal_key_file_name.pem 2048
  ```

  Then, export the RSA public key portion of the pair to a file. The public key becomes the value of `CHEF_AUTOMATE_PIVOTAL_KEY`.

  ```
  openssl rsa -in pivotal_key_file_name.pem -pubout -out public.pem -outform PEM
  ```

- On Windows-based computers, you can use the PuTTYgen utility to generate a base64-encoded RSA key pair. For more information, see PuTTYgen - Key Generator for PuTTY on Windows on SSH.com.

Create a Chef Automate Server in AWS CloudFormation

This section describes how to use an AWS CloudFormation template to build a stack that creates an AWS OpsWorks for Chef Automate server. You can do this by using the AWS CloudFormation console or the AWS CLI. An example AWS CloudFormation template is available for you to use to build an AWS OpsWorks for Chef Automate server stack. Be sure to update the example template with your own server name, IAM roles, instance profile, server description, backup retention count, maintenance options, and optional tags. If your server will use a custom domain, you must specify values for the `CustomDomain`, `CustomCertificate`, and `CustomPrivateKey` parameters in your AWS CloudFormation template. You can specify the `CHEF_AUTOMATE_ADMIN_PASSWORD` and `CHEF_AUTOMATE_PIVOTAL_KEY` engine attributes and their values in the AWS CloudFormation template, or provide just the attributes, and then specify values for the attributes in the AWS CloudFormation **Create Stack** wizard or **create-stack** command. For more information about these attributes, see the section called “Create a Chef Automate server in the AWS Management Console” in the Getting Started section of this guide.

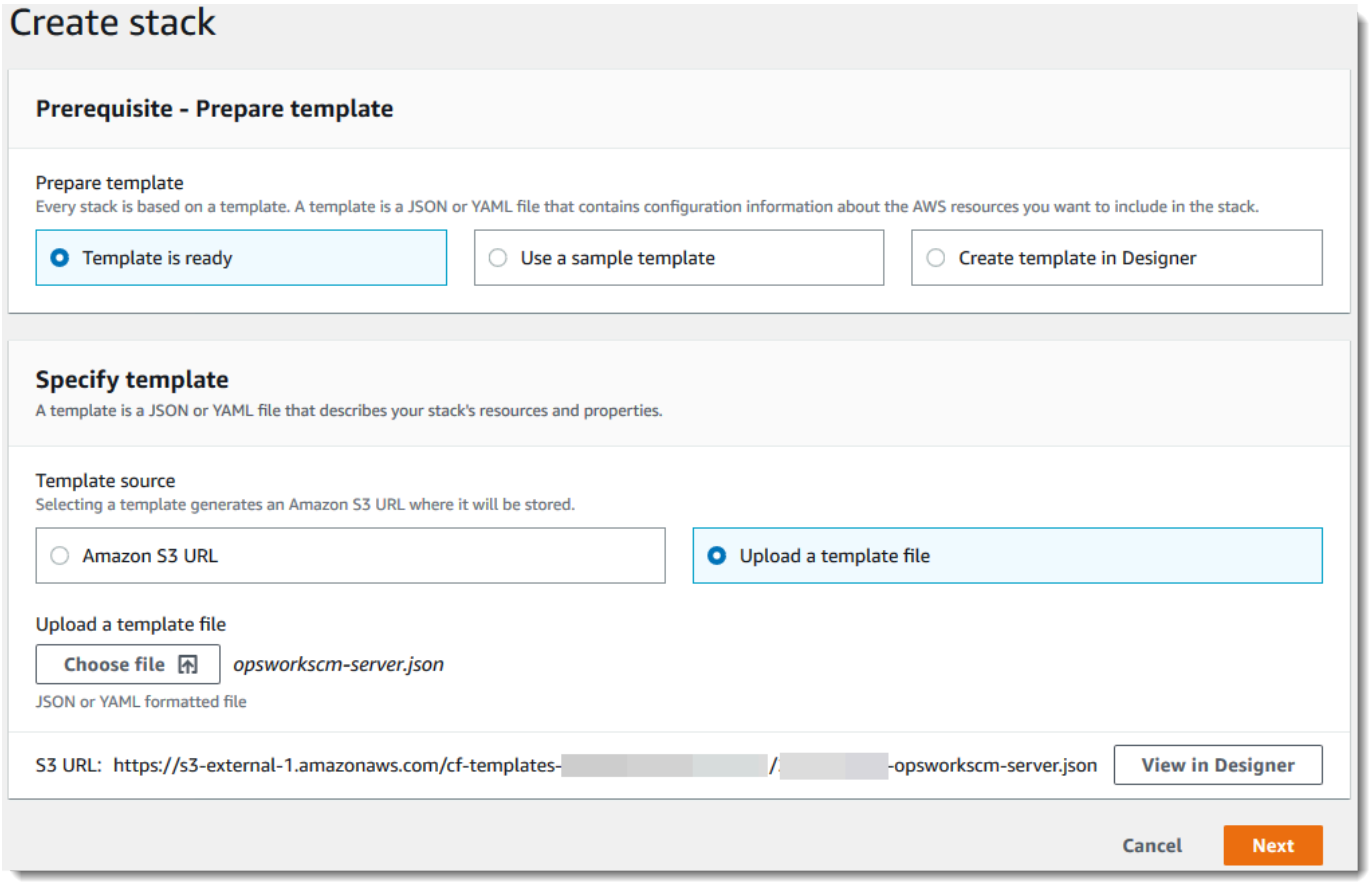
Topics

- Create a Chef Automate Server by using AWS CloudFormation (Console)
- Create a Chef Automate Server by using AWS CloudFormation (CLI)

Create a Chef Automate Server by using AWS CloudFormation (Console)

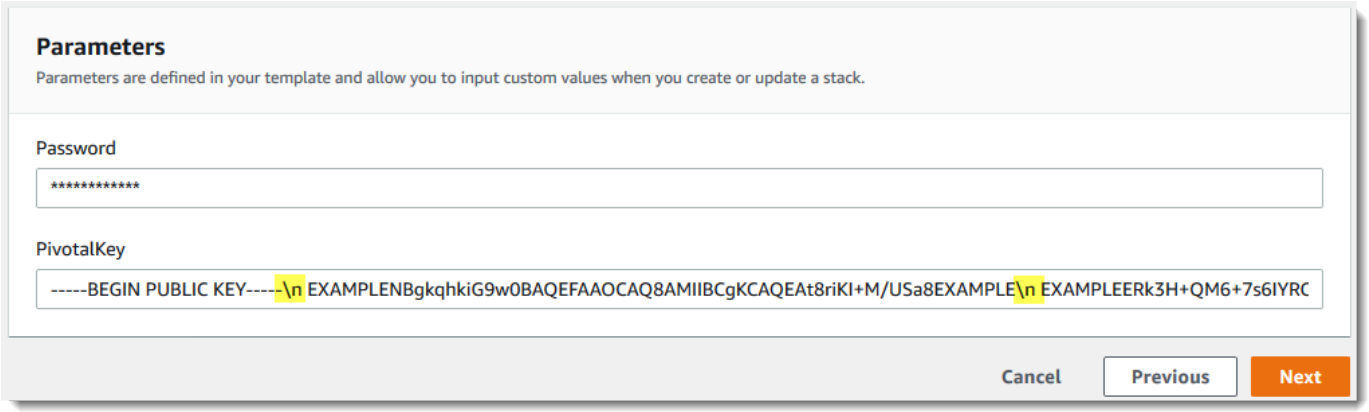
1. Sign in to the AWS Management Console and open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
2. On the AWS CloudFormation home page, choose **Create stack**.
3. In **Prerequisite - Prepare template**, if you are using the example AWS CloudFormation template, choose **Template is ready**.
4. In **Specify template**, choose the source of your template. For this walkthrough, choose **Upload a template file**, and upload an AWS CloudFormation template that creates a Chef Automate server. Browse for your template file, and then choose **Next**.

   An AWS CloudFormation template can be in either YAML or JSON format. An example AWS CloudFormation template is available for you to use; be sure to replace example values with your own. You can use the AWS CloudFormation template designer to build a new template or validate an existing one. For more information about how to do this, see AWS CloudFormation Designer Interface Overview in the *AWS CloudFormation User Guide*.



5. On the **Specify stack details** page, enter a name for your stack. This won't be the same as the name of your server, it is only a stack name. In the **Parameters** area, paste the values that you created in the section called “Prerequisites”. Enter the password in **Password**.

   Paste the contents of the RSA key file in **PivotalKey**. In the AWS CloudFormation console, you must add newline (**\n**) characters at the end of each line of the pivotal key value, as shown in the following screenshot. Choose **Next**.



6. On the **Configure stack options** page, you can add tags to the server you're creating with the stack, and choose an IAM role for creating resources if you have not already specified an IAM role to use in your template. When you're finished specifying options, choose **Next**. For more information about advanced options such as rollback triggers, see Setting AWS CloudFormation Stack Options in the *AWS CloudFormation User Guide*.
7. On the **Review** page, review your choices. When you are ready to create the server stack, choose **Create stack**.

   While you are waiting for AWS CloudFormation to create the stack, view the stack creation status. If stack creation fails, review the error messages shown in the console to help you resolve the issues. For more information about troubleshooting errors in AWS CloudFormation stacks, see Troubleshooting Errors in the *AWS CloudFormation User Guide*.

   When server creation is finished, your Chef Automate server is available on the AWS OpsWorks for Chef Automate home page, with a status of **online**. Generate a new Starter Kit and the Chef Automate dashboard credentials from the server's Properties page. After the server is online, the Chef Automate dashboard is available on the server's domain, at a URL in the following format: `https://your_server_name-randomID.region.opsworks-cm.io`.

> **Note**
>
> If you specified a custom domain, certificate, and private key for your server, create a CNAME entry in your enterprise's DNS management tool that maps your custom domain to the endpoint that AWS OpsWorks for Chef Automate automatically generated for the server. You cannot manage the server or connect to the Chef Automate dashboard for the server until you map the generated endpoint to your custom domain value.
> To get the generated endpoint value, run the following AWS CLI command after your server is online:
>
> ```
> aws opsworks describe-servers --server-name server_name
> ```

Create a Chef Automate Server by using AWS CloudFormation (CLI)

If your local computer is not already running the AWS CLI, download and install the AWS CLI by following installation instructions in the *AWS Command Line Interface User Guide*. This section does not describe all parameters that you can use with the **create-stack** command. For more information about **create-stack** parameters, see **create-stack** in the *AWS CLI Reference*.

1. Be sure to complete the Prerequisites for creating an AWS OpsWorks for Chef Automate server.
2. Create a service role and an instance profile.

   Find and copy the ARNs of service roles in your account.

   ```
   aws iam list-roles --path-prefix "/service-role/" --no-paginate
   ```

   In the results of the `list-roles` command, look for service role and instance profile entries that resemble the following. Make a note of the ARNs of the service role and instance profile, and add them to the AWS CloudFormation template that you are using to create your server stack.
3. Create the AWS OpsWorks for Chef Automate server by running the **create-stack** command again.

- Replace *stack_name* with the name of your stack. This is the name of the AWS CloudFormation stack, not your Chef Automate server. The Chef Automate server name is the value of ServerName in your AWS CloudFormation template.
- Replace *template* with the path to your template file, and the extension *yaml or json* with .yaml or .json as appropriate.
- The values for --parameters correspond to EngineAttributes from the CreateServer API. For Chef, the user-provided engine attributes to create a server are CHEF_AUTOMATE_PIVOTAL_KEY, a base64-encoded RSA public key that you generate by using utilities described in the section called "Prerequisites", and CHEF_AUTOMATE_ADMIN_PASSWORD, a password of between eight and 32 characters that you create. For more information about the CHEF_AUTOMATE_ADMIN_PASSWORD, see Create a Chef Automate server by using the AWS CLI. You can provide a pointer to the PEM file that contains your pivotal key as the value of the PivotalKey parameter, as shown in the example. If the values for CHEF_AUTOMATE_ADMIN_PASSWORD and CHEF_AUTOMATE_PIVOTAL_KEY are not specified in your template, you must provide the values in your AWS CLI command.

```
aws cloudformation create-stack --stack-name stack_name
 --template-body file://template.yaml or json --parameters
 ParameterKey=PivotalKey,ParameterValue="base64_encoded_RSA_public_key_value"
```

The following is an example that includes sample values for the CHEF_AUTOMATE_ADMIN_PASSWORD and CHEF_AUTOMATE_PIVOTAL_KEY attributes. Run a similar command if you did not specify values for these attributes in your AWS CloudFormation template.

```
aws cloudformation create-stack --stack-name "OpsWorksCMChefServerStack"
 --template-body file://opsworkscm-server.yaml --parameters
 ParameterKey=PivotalKey,ParameterValue="$(openssl rsa -in "pivotalKey.pem" -
pubout)" ParameterKey=Password,ParameterValue="SuPer\$ecret890"
```

4. When the stack creation is finished, open the Properties page for the new server in the AWS OpsWorks for Chef Automate console, and download a starter kit. Downloading a new starter kit resets the Chef Automate dashboard administrator password.

5. If your server will use a custom domain, certificate, and private key, follow steps for configuring `knife.rb` in (Optional) Configure `knife` to Work with a Custom Domain, and then go on to step 7.

6. To use `knife` commands on the new server, update Chef `knife.rb` configuration file settings. An example `knife.rb` file is included with the starter kit. The following example shows how to set up `knife.rb` on a server that does not use a custom domain. If you are using a custom domain, see (Optional) Configure `knife` to Work with a Custom Domain for `knife` configuration instructions.

   - Replace *`ENDPOINT`* with the server's endpoint value. This is part of the output of the stack creation operation. You can get the endpoint by running the following command.

     ```
     aws cloudformation describe-stacks --stack-name stack_name
     ```

   - Replace *`key_pair_file.pem`* in the `client_key` configuration with the name of the PEM file that contains the `CHEF_AUTOMATE_PIVOTAL_KEY` that you used to create your server.

     ```
     base_dir = File.join(File.dirname(File.expand_path(__FILE__)), '..')

     log_level                :info
     log_location             STDOUT
     node_name                'pivotal'
     client_key               File.join(base_dir, '.chef', 'key_pair_file.pem')
     syntax_check_cache_path  File.join(base_dir, '.chef', 'syntax_check_cache')
     cookbook_path            [File.join(base_dir, 'cookbooks')]

     chef_server_url          'ENDPOINT/organizations/default'
     ssl_ca_file              File.join(base_dir, '.chef', 'ca_certs', 'opsworks-cm-
     ca-2020-root.pem')
     trusted_certs_dir        File.join(base_dir, '.chef', 'ca_certs')
     ```

7. When the server creation process is finished, go on to the section called “Finish configuration and upload cookbooks”. If stack creation fails, review the error messages shown in the console to help you resolve the issues. For more information about troubleshooting errors in AWS CloudFormation stacks, see Troubleshooting Errors in the *AWS CloudFormation User Guide*.

Update an AWS OpsWorks for Chef Automate Server to Use a Custom Domain

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section describes how to update an existing AWS OpsWorks for Chef Automate server to use a custom domain and certificate by using a backup of the server to create a new server. Essentially, you are copying an existing AWS OpsWorks for Chef Automate 2.0 server by creating a new server from a backup, then configuring the new server to use a custom domain, certificate, and private key.

Topics

- Prerequisites
- Limitations
- Update a Server to Use a Custom Domain
- See Also

Prerequisites

The following are requirements for updating an existing AWS OpsWorks for Chef Automate server to use a custom domain and certificate.

- The server that you want to update (or copy) must be running Chef Automate 2.0.
- Decide which backup you want to use to create a new server. You must have at least one backup available of the server that you want to update. For more information about backups in AWS OpsWorks for Chef Automate, see Back Up an AWS OpsWorks for Chef Automate Server.
- Have ready the service role and instance profile ARNs that you used to create the existing server that is the source of your backup.

- Be sure that you are running the most current release of the AWS CLI. For more information about updating your AWS CLI tools, see Installing the AWS CLI in the *AWS Command Line Interface User Guide*.

Limitations

When you update an existing server by creating a new server from a backup, the new server cannot be exactly the same as the existing AWS OpsWorks for Chef Automate server.

- You can only complete this procedure by using the AWS CLI or one of the AWS SDKs. You cannot create a new server from a backup by using the AWS Management Console.
- The new server cannot use the same name as the existing server within an account, and within an AWS Region. The name must be different from the existing server that you used as the source of the backup.
- Nodes that were attached to the existing server are not managed by the new server. You must do one of the following.
  - Attach different nodes, because nodes cannot be managed by more than one Chef Automate server.
  - Migrate the nodes from the existing server (the source of the backup) to the new server and the new custom domain endpoint. For more information about how to migrate nodes, see in the Chef documentation.

Update a Server to Use a Custom Domain

To update an existing Chef Automate 2.0 server, you make a copy of it by running the `create-server` command, adding parameters to specify a backup, a custom domain, a custom certificate, and a custom private key.

1. If you do not have service role or instance profile ARNs available to specify in your `create-server` command, follow steps 1-5 in Create a Chef Automate server by using the AWS CLI to create a service role and instance profile that you can use.
2. If you have not already done so, find the backup of the existing Chef Automate 2.0 server on which you want to base a new server with a custom domain. Run the following command to show information about all AWS OpsWorks for Chef Automate backups in your account, and in a region. Be sure to note the ID of the backup that you want to use.

```
aws opsworks-cm --region region name describe-backups
```

3. Create the AWS OpsWorks for Chef Automate server by running the `create-server` command.

   - The `--engine` value is `ChefAutomate`, `--engine-model` is `Single`, and `--engine-version` is 12.
   - The server name must be unique within your AWS account, within each region. Server names must start with a letter; then letters, numbers, or hyphens (-) are allowed, up to a maximum of 40 characters.
   - Use the instance profile ARN and service role ARN from step 1.
   - Valid instance types are `m5.large`, `r5.xlarge`, or `r5.2xlarge`. For more information about the specifications of these instance types, see Instance Types in the *Amazon EC2 User Guide*.
   - The `--engine-attributes` parameter is optional; if you don't specify one or both values, the server creation process generates the values for you. If you add `--engine-attributes`, specify either the `CHEF_AUTOMATE_PIVOTAL_KEY` value that you generated in Step 2, a `CHEF_AUTOMATE_ADMIN_PASSWORD`, or both.

     If you do not set a value for `CHEF_AUTOMATE_ADMIN_PASSWORD`, a password is generated and returned as part of the `create-server` response. You can also download the starter kit again in the console, which regenerates this password. The password length is a minimum of eight characters, and a maximum of 32. The password can contain letters, numbers, and special characters (`!/@#$%^+=_`). The password must contain at least one lower case letter, one upper case letter, one number, and one special character.
   - An SSH key pair is optional, but can help you connect to your Chef Automate server if you need to reset the Chef Automate dashboard administrator password. For more information about creating an SSH key pair, see Amazon EC2 Key Pairs in the *Amazon EC2 User Guide*.
   - To use a custom domain, add the following parameters to your command. Otherwise, the Chef Automate server creation process automatically generates an endpoint for you. All three parameters are required to configure a custom domain. For information about additional requirements for using these parameters, see CreateServer in the OpsWorks CM API Reference.
     - `--custom-domain` - An optional public endpoint of a server, such as `https://aws.my-company.com`.

  - `--custom-certificate` - A PEM-formatted HTTPS certificate. The value can be be a single, self-signed certificate, or a certificate chain.
  - `--custom-private-key` - A private key in PEM format for connecting to the server by using HTTPS. The private key must not be encrypted; it cannot be protected by a password or passphrase.
- Weekly system maintenance is required. Valid values must be specified in the following format: `DDD:HH:MM`. The specified time is in coordinated universal time (UTC). If you do not specify a value for `--preferred-maintenance-window`, the default value is a random, one-hour period on Tuesday, Wednesday, or Friday.
- Valid values for `--preferred-backup-window` must be specified in one of the following formats: `HH:MM` for daily backups, or `DDD:HH:MM` for weekly backups. The specified time is in UTC. The default value is a random, daily start time. To opt out of automatic backups, add the parameter `--disable-automated-backup` instead.
- For `--security-group-ids`, enter one or more security group IDs, separated by a space.
- For `--subnet-ids`, enter a subnet ID.
- For `--backup-id`, enter the ID of the backup that you copied in step 2.

```
aws opsworks-cm create-server --engine "ChefAutomate" --engine-model "Single"
 --engine-version "12" --server-name "server_name" --instance-profile-arn
 "instance_profile_ARN" --instance-type "instance_type" --engine-attributes
 '{"CHEF_AUTOMATE_PIVOTAL_KEY":"pivotal_key","CHEF_AUTOMATE_ADMIN_PASSWORD":"password"}'
 --key-pair "key_pair_name" --preferred-maintenance-window
 "ddd:hh:mm" --preferred-backup-window "ddd:hh:mm" --security-group-
ids security_group_id1 security_group_id2 --service-role-arn "service_role_ARN" --
subnet-ids subnet_ID --backup-id backup_ID
```

The following example creates a Chef Automate server that uses a custom domain.

```
aws opsworks-cm create-server --engine "ChefAutomate" --engine-model "Single" --
engine-version "12" \
    --server-name "my-custom-domain-server" \
    --instance-profile-arn "arn:aws:iam::12345678912:instance-profile/aws-opsworks-
cm-ec2-role" \
    --instance-type "m5.large" \
    --engine-attributes
 '{"CHEF_AUTOMATE_PIVOTAL_KEY":"MZZE...Wobg","CHEF_AUTOMATE_ADMIN_PASSWORD":"zZZzDj2DLYXSZF
 \
```

```
    --custom-domain "my-chef-automate-server.my-corp.com" \
    --custom-certificate "-----BEGIN CERTIFICATE----- EXAMPLEqEXAMPLE== -----END
 CERTIFICATE-----" \
    --custom-private-key "-----BEGIN RSA PRIVATE KEY----- EXAMPLEqEXAMPLE= -----END
 RSA PRIVATE KEY-----" \
    --key-pair "amazon-test" \
    --preferred-maintenance-window "Mon:08:00" \
    --preferred-backup-window "Sun:02:00" \
    --security-group-ids sg-b00000001 sg-b0000008 \
    --service-role-arn "arn:aws:iam::12345678912:role/service-role/aws-opsworks-cm-
service-role" \
    --subnet-ids subnet-300aaa00 \
    --backup-id MyChefServer-20191004122143125
```

4. AWS OpsWorks for Chef Automate takes about 15 minutes to create a new server. In the output of the `create-server` command, copy the value of the `Endpoint` attribute. The following is an example.

   ```
   "Endpoint": "automate-07-exampleexample.opsworks-cm.us-east-1.amazonaws.com"
   ```

   Do not dismiss the output of the `create-server` command or close your shell session, because the output can contain important information that is not shown again. To get passwords and the starter kit from the `create-server` results, go on to the next step.

5. If you opted to have AWS OpsWorks for Chef Automate generate a key and password for you, you can extract them in usable formats from the `create-server` results by using a JSON processor such as jq. After you install jq, you can run the following commands to extract the pivotal key, Chef Automate dashboard administrator password, and starter kit. If you did not provide your own pivotal key and password in Step 3, be sure to save the extracted pivotal key and administrator password in convenient but secure locations.

   ```
   #Get the Chef password:
   cat resp.json | jq -r '.Server.EngineAttributes[] | select(.Name ==
    "CHEF_AUTOMATE_ADMIN_PASSWORD") | .Value'

   #Get the Chef Pivotal Key:
   cat resp.json | jq -r '.Server.EngineAttributes[] | select(.Name ==
    "CHEF_AUTOMATE_PIVOTAL_KEY") | .Value'

   #Get the Chef Starter Kit:
   ```

```
cat resp.json | jq -r '.Server.EngineAttributes[] | select(.Name ==
 "CHEF_STARTER_KIT") | .Value' | base64 -D > starterkit.zip
```

6. Optionally, if you did not extract the starter kit from `create-server` command results, you can download a new starter kit from the server's Properties page in the AWS OpsWorks for Chef Automate console. Downloading a new starter kit resets the Chef Automate dashboard administrator password.
7. Create a CNAME entry in your enterprise's DNS management tool to point your custom domain to the AWS OpsWorks for Chef Automate endpoint that you copied in step 4. You cannot reach or sign in to the server until you complete this step.
8. When the server creation process is finished, go on to the section called “Finish configuration and upload cookbooks”.

See Also

- Create a Chef Automate server by using the AWS CLI
- Restore an AWS OpsWorks for Chef Automate Server from a Backup
- CreateServer in the OpsWorks CM API Reference
- **create-server** in the *AWS CLI Command Reference*

Regenerate the starter kit for an AWS OpsWorks for Chef Automate server

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The starter kit for AWS OpsWorks for Chef Automate contains a README file with examples, a `knife.rb` configuration file, and a private key for the primary, or pivotal, user. A new key pair is generated—and the old key is reset—each time you download the starter kit. You can regenerate the starter kit for an AWS OpsWorks for Chef Automate server in one of two ways:

- In the OpsWorks console, on the **Actions** menu of the details page for an AWS OpsWorks for Chef Automate server. You are prompted to confirm whether you want to regenerate and reset the old pivotal key.
- By running commands in the AWS CLI.

For more information about how to use the starter kit, see Configure the Chef Server Using the Starter Kit.

Regenerate the AWS OpsWorks for Chef Automate starter kit with the AWS CLI

> **Note**
>
> When you regenerate the starter kit, you also regenerate and reset the authentication key pair for your Chef Automate server, and delete the current key pair.

Regenerate the starter kit by running the **update-server-engine-attributes** command. In an AWS CLI session, run the following command. Specify your server name as the value of `--server-name`. To set a public key of your own as the value of `CHEF_AUTOMATE_PIVOTAL_KEY`, specify the value of the public key in `--attribute-value`. Otherwise, set `--attribute-value` to null.

```
aws opsworks-cm update-server-engine-attributes \
   --server-name server_name \
   --attribute-name "CHEF_AUTOMATE_PIVOTAL_KEY" \
   --attribute-value your_public_key
```

The following command is an example that specifies a public key value that the server's administrator wants to use.

```
aws opsworks-cm update-server-engine-attributes \
   --server-name your-test-server \
   --attribute-name "CHEF_AUTOMATE_PIVOTAL_KEY" \
   --attribute-value "-----BEGIN PUBLIC KEY-----ExamplePublicKey-----END PUBLIC
 KEY-----"
```

The following command is an example that lets AWS OpsWorks for Chef Automate regenerate the public key.

```
aws opsworks-cm update-server-engine-attributes \
   --server-name your-test-server \
   --attribute-name "CHEF_AUTOMATE_PIVOTAL_KEY" \
   --attribute-value null
```

The output of this command is information about the server, and a base64-encoded ZIP file. The ZIP file contains a Chef starter kit, which includes a README, a configuration file, and the required RSA private key. Save this file, unzip it, and then change to the directory where you've unzipped the file contents. From this directory, you can run `knife` commands.

Working with Tags on AWS OpsWorks for Chef Automate Resources

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Tags are words or phrases that act as metadata for identifying and organizing your AWS resources. In AWS OpsWorks for Chef Automate, a resource can have up to 50 user-applied tags. Each tag consists of a key and one optional value. You can apply tags to the following resources in AWS OpsWorks for Chef Automate:

- AWS OpsWorks for Chef Automate servers
- Backups of AWS OpsWorks for Chef Automate servers

Tags on AWS resources can help you track costs, control access to resources, group resources for automating tasks, or organize resources by purpose or lifecycle stage. For more information about the benefits of tags, see AWS Tagging Strategies in AWS Answers and Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*.

To use tags to control access to AWS OpsWorks for Chef Automate servers or backups, you create or edit policy statements in AWS Identity and Access Management (IAM). For more information,

see Controlling Access to AWS Resources Using Resource Tags in the *AWS Identity and Access Management User Guide*.

When you apply tags to an AWS OpsWorks for Chef Automate server, the tags are also applied to the server's backups, the Amazon S3 bucket that stores the backups, the server's Amazon EC2 instance, secrets for the server that are stored in AWS Secrets Manager, and the Elastic IP address used by the server. Tags are not propagated to the AWS CloudFormation stack that OpsWorks uses to create your server.

Topics

- How Tags Work in AWS OpsWorks for Chef Automate
- Add and Manage Tags in AWS OpsWorks for Chef Automate (Console)
- Add and Manage Tags in AWS OpsWorks for Chef Automate (CLI)
- See Also

How Tags Work in AWS OpsWorks for Chef Automate

In this release, you can add and manage tags by using the OpsWorks CM API or the AWS Management Console. OpsWorks CM also attempts to add tags that you add to a server to the AWS resources that are associated with the server, including the EC2 instance, secrets in Secrets Manager, Elastic IP address, security group, S3 bucket, and backups. The following table provides an overview of how you add and manage tags in AWS OpsWorks for Chef Automate.

Action	What to use
Add tags to a new AWS OpsWorks for Chef Automate server or a backup that you are creating manually.	• Choose **Create Chef Automate server** and add tags on the **Configure advanced settings** page. • Choose **Create backup** on the **Backups** page for an existing server, and add tags on the **Create a backup of your Chef Automate 2 server** page. • Add a `Tags` parameter to the `CreateServer` or `CreateBackup` commands.

Action	What to use
View tags on a resource.	• On the details page for your server, choose **Tags** in the navigation pane. • On the **Backups** page for your server, select a backup, and then choose **Edit backup**. • Run the `ListTagsForResource` command.
Add tags to an existing AWS OpsWorks for Chef Automate server or a backup, regardless of whether the backup was created manually or automatically.	• On the details page for your server, choose **Tags** in the navigation pane, and then choose **Edit**. • On the **Backups** page for your server, select a backup, and then choose **Edit backup**. • Run the `TagResource` command.
Delete tags from a resource.	• On the details page for your server, choose **Tags** in the navigation pane, and then choose **Edit**. Choose **X** next to tags that you want to delete. • On the **Backups** page for your server, select a backup, and then choose **Edit backup**. Choose **X** next to tags that you want to delete. • Run the `UntagResource` command.

`DescribeServers` and `DescribeBackups` responses do not include tag information. To show tags, use the `ListTagsForResource` API.

Add and Manage Tags in AWS OpsWorks for Chef Automate (Console)

Procedures in this section are performed in the AWS Management Console.

If you add tags, a tag key cannot be empty. The key can be a maximum of 127 characters, and can contain only Unicode letters, numbers, or separators, or the following special characters: `+ - = . _ : / @` A tag value is optional. You can add a tag that has a key, but no value. The value can be a

maximum of 255 characters, and can contain only Unicode letters, numbers, or separators, or the following special characters: `+ - = . _ : / @`

Topics

- Add Tags to a New AWS OpsWorks for Chef Automate Server (Console)
- Add Tags to a New Backup (Console)
- Add or View Tags on an Existing Server (Console)
- Add or View Tags on an Existing Backup (Console)
- Delete Tags from a Server (Console)
- Delete Tags from a Backup (Console)

Add Tags to a New AWS OpsWorks for Chef Automate Server (Console)

1. Be sure to complete any prerequisites for creating an AWS OpsWorks for Chef Automate server.
2. Follow steps 1-10 in Create a Chef Automate Server.
3. After you specify automated backup settings, add tags in the **Tags** area of the **Configure advanced settings** page. You can add a maximum of 50 tags. When you are finished adding tags, choose **Next**.
4. Go on to step 13 of Create a Chef Automate Server, and review settings you have chosen for the new server.

Add Tags to a New Backup (Console)

1. On the AWS OpsWorks for Chef Automate home page, choose an existing Chef Automate server.
2. From the server's details page, choose **Backups** in the navigation pane.
3. On the **Backups** page, choose **Create backup**.
4. Add tags. Choose **Create** when you are finished adding tags.

Add or View Tags on an Existing Server (Console)

1. On the AWS OpsWorks for Chef Automate home page, choose an existing Chef Automate server to open its details page.

2. Choose **Tags** in the navigation pane, or at the bottom of the details page, choose **View all tags**.
3. On the **Tags** page, choose **Edit**.
4. Add or edit tags on the server. Choose **Save** when you are finished.

> **Note**
>
> Be aware that changing tags on your Chef Automate server also changes tags on resources that are associated with the server, such as the EC2 instance, Elastic IP address, security group, S3 bucket, and backups.

Add or View Tags on an Existing Backup (Console)

1. On the AWS OpsWorks for Chef Automate home page, choose an existing Chef Automate server to open its details page.
2. Choose **Backups** in the navigation pane, or in the **Recent backups** area of the details page, choose **View all backups**.
3. On the **Backups** page, choose a backup to manage, and then choose **Edit backup**.
4. Add or edit tags on the backup. Choose **Update** when you are finished.

Delete Tags from a Server (Console)

1. On the AWS OpsWorks for Chef Automate home page, choose an existing Chef Automate server to open its details page.
2. Choose **Tags** in the navigation pane, or at the bottom of the details page, choose **View all tags**.
3. On the **Tags** page, choose **Edit**.
4. Choose **X** next to a tag to delete the tag. Choose **Save** when you are finished.

> **Note**
>
> Be aware that changing tags on your Chef Automate server also changes tags on resources that are associated with the server, such as the EC2 instance, Elastic IP address, security group, S3 bucket, and backups.

Delete Tags from a Backup (Console)

1. On the AWS OpsWorks for Chef Automate home page, choose an existing Chef Automate server to open its details page.
2. Choose **Backups** in the navigation pane, or in the **Recent backups** area of the details page, choose **View all backups**.
3. On the **Backups** page, choose a backup to manage, and then choose **Edit backup**.
4. Choose **X** next to a tag to delete the tag. Choose **Update** when you are finished.

Add and Manage Tags in AWS OpsWorks for Chef Automate (CLI)

Procedures in this section are performed in the AWS CLI. Be sure that you are running the latest release of the AWS CLI before you start working with tags. For more information about installing or updating the AWS CLI, see Installing the AWS CLI in the *AWS Command Line Interface User Guide*.

If you add tags, a tag key cannot be empty. The key can be a maximum of 127 characters, and can contain only Unicode letters, numbers, or separators, or the following special characters: `+ - = . _ : / @` A tag value is optional. You can add a tag that has a key, but no value. The value can be a maximum of 255 characters, and can contain only Unicode letters, numbers, or separators, or the following special characters: `+ - = . _ : / @`

Topics

- Add Tags to a New AWS OpsWorks for Chef Automate Server (CLI)
- Add Tags to a New Backup (CLI)
- Add Tags to Existing Servers or Backups (CLI)
- List Resource Tags
- Delete Tags from a Resource

Add Tags to a New AWS OpsWorks for Chef Automate Server (CLI)

You can use the AWS CLI to add tags when you create an AWS OpsWorks for Chef Automate server. This procedure does not describe in full how to create a server. For detailed information about how to create an AWS OpsWorks for Chef Automate server by using the AWS CLI see Create a Chef Automate server by using the AWS CLI in this guide. You can add up to 50 tags to a server.

1. Be sure to complete any prerequisites for creating an AWS OpsWorks for Chef Automate server.
2. Complete steps 1-5 of Create a Chef Automate server by using the AWS CLI.
3. For step 6, when you run the `create-server` command, add the `--tags` parameter to the command, as shown in the following example.

   ```
   aws opsworks-cm create-server ... --tags Key=Key1,Value=Value1
    Key=Key2,Value=Value2
   ```

   The following is an example showing only the tags portion of the `create-server` command.

   ```
   aws opsworks-cm create-server ... --tags Key=Stage,Value=Production
    Key=Department,Value=Marketing
   ```

4. Complete the remaining steps in Create a Chef Automate server by using the AWS CLI. To verify that your tags were added to the new server, follow steps in List Resource Tags in this topic.

Add Tags to a New Backup (CLI)

You can use the AWS CLI to add tags when you create a new, manual backup of an AWS OpsWorks for Chef Automate server. This procedure does not describe in full how to create a manual backup. For detailed information about how to create a manual backup, see "To perform a manual backup in the AWS CLI" in Back Up an AWS OpsWorks for Chef Automate Server. You can add up to 50 tags to a backup. If a server has tags, new backups are automatically tagged with the server's tags.

By default, when you create a new AWS OpsWorks for Chef Automate server, automated backups are enabled. You can add tags to an automated backup by running the `tag-resource` command, described in Add Tags to Existing Servers or Backups (CLI) in this topic.

- To add tags to a manual backup as you're creating the backup, run the following command. Only the tags portion of the command is shown. For an example of the full `create-backup` command, see "To perform a manual backup in the AWS CLI" in Back Up an AWS OpsWorks for Chef Automate Server.

  ```
  aws opsworks-cm create-backup ... --tags Key=Key1,Value=Value1
   Key=Key2,Value=Value2
  ```

The following example shows only the tags portion of the `create-backup` command.

```
aws opsworks-cm create-backup ... --tags Key=Stage,Value=Production
 Key=Department,Value=Marketing
```

Add Tags to Existing Servers or Backups (CLI)

You can run the `tag-resource` command to add tags to existing AWS OpsWorks for Chef Automate servers or backups (whether the backups were created automatically or manually). Specify the Amazon Resource Number (ARN) of a target resource to add tags to it.

1. To get the ARN of the resource to which you want to apply tags:

   - For a server, run `describe-servers --server-name` *`server_name`*. The results of the command show the server ARN.
   - For a backup, run `describe-backups --backup-id` *`backup_ID`*. The results of the command show the backup ARN. You can also run `describe-backups --server-name` *`server_name`* to show information about all backups for a specific AWS OpsWorks for Chef Automate server.

   The following example shows only the `ServerArn` in results of a `describe-servers --server-name opsworks-cm-test` command. The `ServerArn` value is added to a `tag-resource` command to add tags to the server.

```
{
    "Servers": [
        {
            ...
            "ServerArn": "arn:aws:opsworks-cm:us-west-2:123456789012:server/
opsworks-cm-test/EXAMPLEd-66b0-4196-8274-d1a2bEXAMPLE"
        }
    ]
}
```

2. Run the `tag-resource` command with the ARN that you returned in step 1.

```
aws opsworks-cm tag-resource --resource-arn "server_or_backup_ARN" --tags
 Key=Key1,Value=Value1 Key=Key2,Value=Value2
```

The following is an example.

```
aws opsworks-cm tag-resource --resource-arn "arn:aws:opsworks-cm:us-
west-2:123456789012:server/opsworks-cm-test/EXAMPLEd-66b0-4196-8274-d1a2bEXAMPLE"
 --tags Key=Stage,Value=Production Key=Department,Value=Marketing
```

3. To verify that tags were added successfully, go on to the next procedure, List Resource Tags.

List Resource Tags

You can run the `list-tags-for-resource` command to show the tags that are attached to AWS OpsWorks for Chef Automate servers or backups. Specify the ARN of a target resource to view its tags.

1. To get the ARN of the resource for which you want to list tags:
   - For a server, run `describe-servers --server-name` *server_name*. The results of the command show the server ARN.
   - For a backup, run `describe-backups --backup-id` *backup_ID*. The results of the command show the backup ARN. You can also run `describe-backups --server-name` *server_name* to show information about all backups for a specific AWS OpsWorks for Chef Automate server.
2. Run the `list-tags-for-resource` command with the ARN that you returned in step 1.

```
aws opsworks-cm list-tags-for-resource --resource-arn "server_or_backup_ARN"
```

The following is an example.

```
aws opsworks-cm tag-resource --resource-arn "arn:aws:opsworks-cm:us-
west-2:123456789012:server/opsworks-cm-test/EXAMPLEd-66b0-4196-8274-d1a2bEXAMPLE"
```

If there are tags on the resource, the command returns results like the following.

```
{
```

```
    "Tags": [
        {
            "Key": "Stage",
            "Value": "Production"
        },
        {
            "Key": "Department",
            "Value": "Marketing"
        }
    ]
}
```

Delete Tags from a Resource

You can run the `untag-resource` command to delete tags from AWS OpsWorks for Chef Automate servers or backups. If the resource is deleted, the tags on the resource are also deleted. Specify the Amazon Resource Number (ARN) of a target resource to remove tags from it.

1. To get the ARN of the resource from which you want to remove tags:

   - For a server, run `describe-servers --server-name` *`server_name`*. The results of the command show the server ARN.
   - For a backup, run `describe-backups --backup-id` *`backup_ID`*. The results of the command show the backup ARN. You can also run `describe-backups --server-name` *`server_name`* to show information about all backups for a specific AWS OpsWorks for Chef Automate server.

2. Run the `untag-resource` command with the ARN that you returned in step 1. Specify only the tags that you want to delete.

   ```
   aws opsworks-cm untag-resource --resource-arn "server_or_backup_ARN" --tags
    Key=Key1,Value=Value1 Key=Key2,Value=Value2
   ```

   In this example, the `untag-resource` command removes only the tag with a key of `Stage` and a value of `Production`.

   ```
   aws opsworks-cm untag-resource --resource-arn "arn:aws:opsworks-cm:us-
   west-2:123456789012:server/opsworks-cm-test/EXAMPLEd-66b0-4196-8274-d1a2bEXAMPLE"
    --tags Key=Stage,Value=Production
   ```

3. To verify that tags were deleted successfully, follow steps in List Resource Tags in this topic.

See Also

- Create a Chef Automate server by using the AWS CLI
- Back Up an AWS OpsWorks for Chef Automate Server
- AWS Tagging Strategies
- Controlling Access to AWS Resources Using Resource Tags in the *AWS Identity and Access Management User Guide*
- Using Cost Allocation Tags in the *AWS Billing and Cost Management User Guide*
- CreateBackup in the *OpsWorks CM API Reference*
- CreateServer in the *OpsWorks CM API Reference*
- TagResource in the *OpsWorks CM API Reference*
- ListTagsForResource in the *OpsWorks CM API Reference*
- UntagResource in the *OpsWorks CM API Reference*

Back Up and Restore an AWS OpsWorks for Chef Automate Server

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section describes how to back up and restore an AWS OpsWorks for Chef Automate server, and how to delete backups.

Topics

- Back Up an AWS OpsWorks for Chef Automate Server
- Restore an AWS OpsWorks for Chef Automate Server from a Backup

Back Up an AWS OpsWorks for Chef Automate Server

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can define a daily or weekly recurring AWS OpsWorks for Chef Automate server backup, and have the service store the backups in Amazon Simple Storage Service (Amazon S3) on your behalf. Alternatively, you can make manual backups on demand.

Because backups are stored in Amazon S3, they incur additional fees. You can define a backup retention period of up to 30 generations. You can submit a service request to have that limit changed by using AWS support channels. Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

You can add tags to backups of an AWS OpsWorks for Chef Automate server. If you have added tags to an AWS OpsWorks for Chef Automate server, automated backups of the server inherit those tags. For more information about how to add and manage tags on backups, see Working with Tags on AWS OpsWorks for Chef Automate Resources in this guide.

Topics

- Automated Backups
- Manual Backups
- Delete backups

Automated Backups

When you configure your AWS OpsWorks for Chef Automate server, you choose either automated or manual backups. AWS OpsWorks for Chef Automate starts automated backups during the hour and on the day that you choose in the **Automated backup** section of the **Configure advanced settings** page of Setup. After your server is online, you can change backup settings by performing

the following steps, either from the server's tile on the Chef Automate servers home page, or on the server's Properties page.

To change automated backup settings

1. In the **Actions** menu of the server's tile on the **Chef servers** home page, choose **Change settings**
2. To turn off automated backups, choose **No** for the **Enable automated backups** option. Save your changes; you do not need to go on to the next step.
3. In the **Automated Backup** section, change the frequency, start time, or generations to keep. Save your changes.

Manual Backups

You can start a manual backup at any time in the AWS Management Console, or by running the AWS CLI create-backup command. Manual backups are not included in the maximum 30 generations of automated backups that are stored; a maximum of 10 manual backups are stored, and must be manually deleted from Amazon S3.

You can add tags when you create a new, manual backup of an AWS OpsWorks for Chef Automate server. For more information about how to add tags when you create a manual backup, see Add Tags to a New Backup (CLI).

To perform a manual backup in the AWS Management Console

1. On the **Chef Automate servers** page, choose the server that you want to back up.
2. On the properties page for the server, in the left navigation pane, choose **Backups**.
3. Choose **Create backup**.
4. The manual backup is finished when the page shows a green check mark in the backup's **Status** column.

To perform a manual backup in the AWS CLI

- To start a manual backup, run the following AWS CLI command.

```
aws opsworks-cm --region region name create-backup --server-name "Chef server name"
 --description "optional descriptive string"
```

Delete backups

Deleting a backup permanently deletes it from the S3 bucket in which backups are stored.

To delete a backup in the AWS Management Console

1. On the **Chef Automate servers** page, choose the server that you want to back up.
2. On the properties page for the server, in the left navigation pane, choose **Backups**.
3. Choose the backup that you want to delete, and then choose **Delete backup**. You can select only one backup at a time.
4. When you are prompted to confirm the deletion, fill the check box for **Delete the backup, which is stored in an S3 bucket**, and then choose **Yes, Delete**.

To delete a backup in the AWS CLI

- To delete a backup, run the following AWS CLI command, replacing `--backup-id` with the ID of the backup that you want to delete. Backup IDs are in the format *`ServerName-yyyyMMddHHmmssSSS`*. For example, **`test-chef-server-20171218132604388`**.

```
aws opsworks-cm --region region name delete-backup --backup-id ServerName-
yyyyMMddHHmmssSSS
```

Restore an AWS OpsWorks for Chef Automate Server from a Backup

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

After browsing through your available backups, you can choose a point in time from which to restore your AWS OpsWorks for Chef Automate server. Server backups contain only configuration-management software persistent data (cookbooks, registered nodes, etc.). Performing an in-place restoration of a server (that is, restoring the existing AWS OpsWorks for Chef Automate server

to a new EC2 instance) reregisters nodes that were registered at the time of the backup that you use to restore the server, and switches traffic to the new instance if restoration is successful, and the restored AWS OpsWorks for Chef Automate server state is `Healthy`. Restoring to a newly-created AWS OpsWorks for Chef Automate server does not maintain node connections. Restoring a server does not update minor versions of Chef software; it applies the same Chef versions and configuration-management data that are available in the backup that you choose.

Restoring a server typically takes more time than creating a new server; the time depends on the size of the backup you choose. After restoration is complete, the old EC2 instance remains in a `Running` or `Stopped` state, but only temporarily. It is eventually terminated.

In this release, you can use the AWS CLI to restore a Chef server in AWS OpsWorks for Chef Automate.

> **Note**
>
> You can also run the restore-server command to change the current instance type, or to restore or set your SSH key if it is lost or compromised.

To restore a server from a backup

1. In the AWS CLI, run the following command to return a list of available backups and their IDs. Make a note of the ID of the backup that you want to use. Backup IDs are in the format *myServerName-yyyyMMddHHmmssSSS*.

   ```
   aws opsworks-cm --region region name describe-backups
   ```

2. Run the following command.

   ```
   aws opsworks-cm --region region name restore-server --backup-id "myServerName-
   yyyyMMddHHmmssSSS" --instance-type "Type of instance" --key-pair "name of your EC2
    key pair" --server-name "name of Chef server"
   ```

   The following is an example.

   ```
   aws opsworks-cm --region us-west-2 restore-server --backup-id
    "MyChefServer-20161120122143125" --server-name "MyChefServer"
   ```

3. Wait until restoration is complete.

System Maintenance in AWS OpsWorks for Chef Automate

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Mandatory system maintenance ensures that the latest minor versions of Chef Server and Chef Automate Server, including security updates, are always running on an AWS OpsWorks for Chef Automate server. System maintenance is required a minimum of once a week. By using the AWS CLI, you can configure daily automatic maintenance, if desired. You can also use the AWS CLI to perform system maintenance on demand, in addition to scheduled system maintenance.

When new minor versions of Chef software become available, system maintenance is designed to update the minor version of Chef Automate and Chef Server on the server automatically, as soon as it passes AWS testing. AWS performs extensive testing to verify that Chef upgrades are production-ready and do not disrupt existing customer environments, so there can be lags between Chef software releases and their availability for application to existing OpsWorks for Chef Automate servers. To update available minor versions of Chef software on demand, see Starting system maintenance on demand in this topic.

System maintenance launches a new instance from a backup that is performed as part of the maintenance process, which helps reduce risk from degraded or impaired Amazon EC2 instances that undergo periodic maintenance.

> **Important**
>
> System maintenance deletes any files or custom configuration that you have added to the AWS OpsWorks for Chef Automate server. For more information about how to repair configuration or file loss, see Restoring custom configurations and files after maintenance in this topic.

Topics

- Ensuring nodes trust the OpsWorks Certification Authority

- Configuring system maintenance
- Starting system maintenance on demand
- Restoring custom configurations and files after maintenance

Ensuring nodes trust the OpsWorks Certification Authority

> **Note**
>
> Steps in this section are not required if you are using a custom domain and certificate with your AWS OpsWorks for Chef Automate server.

Nodes that you are managing with an AWS OpsWorks for Chef Automate server must authenticate with the server by using certificates. During system maintenance, OpsWorks replaces the server instance, and regenerates new certificates through the OpsWorks certificate authority (CA). To restore communication automatically with your managed nodes after maintenance is finished, nodes should trust the OpsWorks CA that ships with the starter kit, and is hosted in the regions that are supported by AWS OpsWorks for Chef Automate. When you use the OpsWorks CA to establish the trust between nodes and server, nodes reconnect to the new server instance after maintenance. If you are adding EC2 nodes by using the EC2 `userdata` script described in Add nodes automatically in AWS OpsWorks for Chef Automate, nodes are already configured to trust the OpsWorks CA.

- For Linux-based nodes, the S3 bucket location of the CA is `https://opsworks-cm-${REGION}-prod-default-assets.s3.amazonaws.com/misc/opsworks-cm-ca-2020-root.pem`. The OpsWorks trusted CA must be stored in the path `/etc/chef/opsworks-cm-ca-2020-root.pem`.
- For Windows-based nodes, the S3 bucket location of the CA is `https://opsworks-cm-$env:AWS_REGION-prod-default-assets.s3.amazonaws.com/misc/opsworks-cm-ca-2020-root.pem`. The OpsWorks CA must be stored in the root Chef folder; for example, `C:\chef\opsworks-cm-ca-2020-root.pem`

In both paths, the region variable resolves to one of the following.

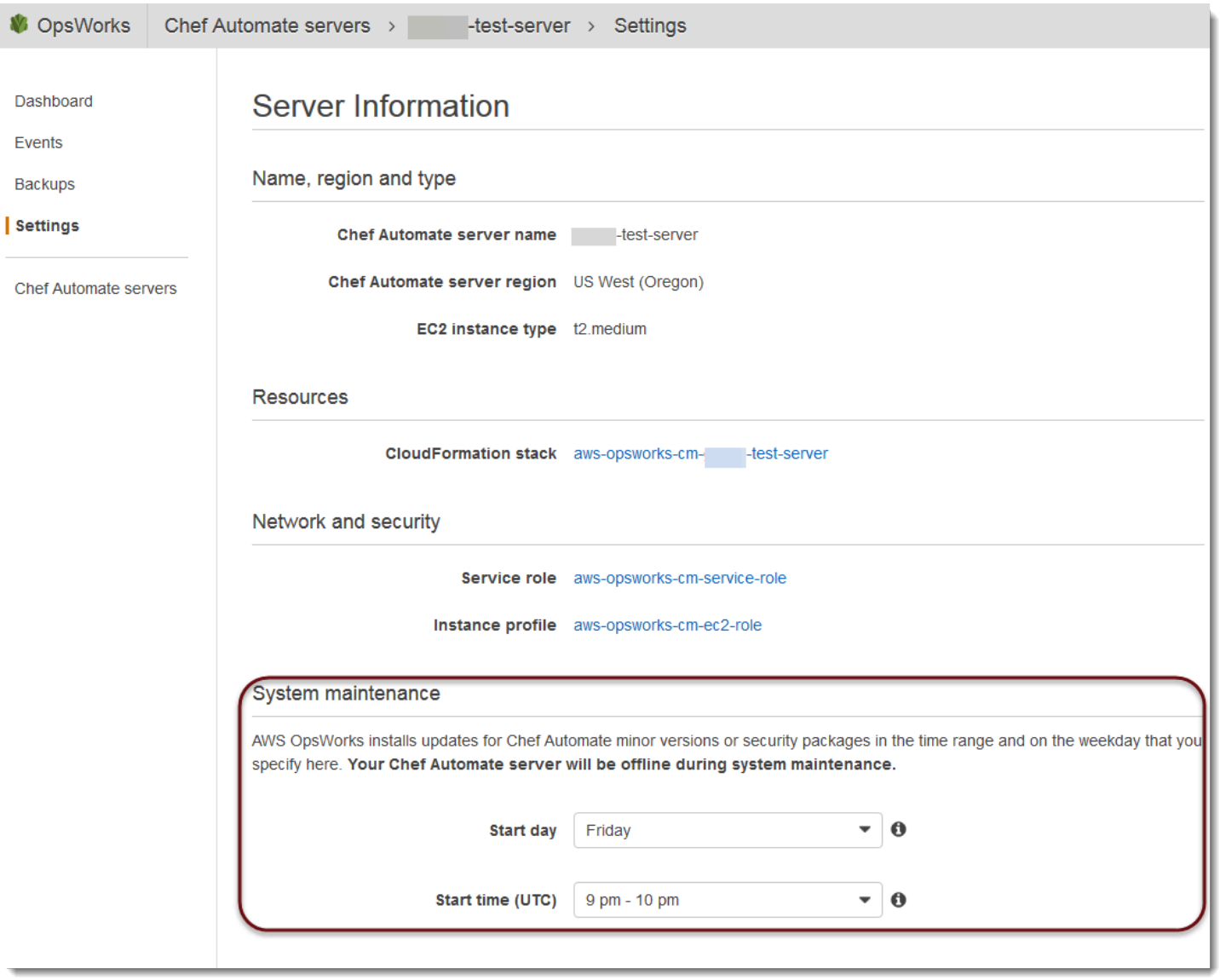
- `us-east-2`
- `us-east-1`

- us-west-1
- us-west-2
- ap-northeast-1
- ap-southeast-1
- ap-southeast-2
- eu-central-1
- eu-west-1

Configuring system maintenance

When you create a new AWS OpsWorks for Chef Automate server, you can configure a weekday and time, in Coordinated Universal Time (UTC), for system maintenance to start. Maintenance starts during the hour that you specify. Because you should expect the server to be offline during system maintenance, choose a time of low server demand within regular office hours. The server status is `UNDER_MAINTENANCE` while maintenance is in progress.

You can also change the system maintenance settings on an existing AWS OpsWorks for Chef Automate server, by changing settings in the **System maintenance** area of the **Settings** page for your server, as shown in the following screenshot.



In the **System maintenance** section, set the day and hour that you want system maintenance to begin.

Configuring system maintenance by using the AWS CLI

You can also configure the system maintenance automatic start time by using the AWS CLI. The AWS CLI lets you configure daily automatic maintenance, if desired, by omitting the three-character weekday prefix.

In a `create-server` command, add the `--preferred-maintenance-window` parameter to your command, after specifying the requirements for creating the server instance (such as instance type, instance profile ARN, and service role ARN). In the following `create-server` example, `--`

`preferred-maintenance-window` is set to `Mon:08:00`, meaning that you've set maintenance to start every Monday morning at 8:00 a.m. UTC.

```
aws opsworks-cm create-server --engine "Chef" --engine-model "Single" --
engine-version "12" --server-name "automate-06" --instance-profile-arn
 "arn:aws:iam::1019881987024:instance-profile/aws-opsworks-cm-ec2-role"
 --instance-type "t2.medium" --key-pair "amazon-test" --service-role-arn
 "arn:aws:iam::044726508045:role/aws-opsworks-cm-service-role" --preferred-maintenance-
window "Mon:08:00"
```

In an `update-server` command, you can update the `--preferred-maintenance-window` value alone, if desired. In the following example, the maintenance window is set to Friday night at 6:15 p.m. UTC.

```
aws opsworks-cm update-server --server-name "shiny-kitchen" --preferred-maintenance-
window "Fri:18:15"
```

To change the start time of the maintenance window to 6:15 p.m. UTC every day, omit the three-character weekday prefix, as shown in the following example.

```
aws opsworks-cm update-server --server-name "shiny-kitchen" --preferred-maintenance-
window "18:15"
```

For more information about setting the preferred system maintenance window by using the AWS CLI, see create-server and update-server.

Starting system maintenance on demand

To start system maintenance on demand, outside of your configured weekly or daily automatic maintenance, run the following AWS CLI command. You cannot start on-demand maintenance in the AWS Management Console.

```
aws opsworks-cm start-maintenance --server-name server_name
```

For more information about this command, see start-maintenance.

Restoring custom configurations and files after maintenance

System maintenance can delete or change custom files or configurations that you have added to your AWS OpsWorks for Chef Automate server.

If, after a maintenance run, your Chef server is missing files or settings that you added by using RunCommand or SSH, you can use an Amazon Machine Image (AMI) to launch a new Amazon EC2 instance. AMIs are available that are built from a server's pre-maintenance configuration.

The new instance is in the same state that the Chef server was before maintenance, and should include your missing files and settings.

> **Important**
>
> You cannot use the new instance to restore your server; the instance cannot be run as a Chef server. You can use the instance only to recover your files and configuration settings.

To launch an EC2 instance from an AMI, in the Amazon EC2 console, open the **Launch** wizard, choose **My AMIs**, and then choose the AMI that has your server name. Follow Amazon EC2 wizard steps as you would for any other instance launch.

Compliance Scans in AWS OpsWorks for Chef Automate

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Compliance scans let you track the compliance of managed nodes in your infrastructure based on predefined policies, also called *rules*. Compliance views let you regularly audit your applications for vulnerabilities and noncompliant configurations. Chef offers over 100 predefined compliance *profiles*—collections of rules that apply to specific node configurations—that you can use in your compliance scans. You can also use the Chef InSpec language to create your own custom profiles.

If your server is not yet running Chef Automate 2.0, you can set up Chef Compliance manually, by installing the Audit cookbook.

> **Note**
>
> The minimum supported version of the Chef Infra client agent software (`chef-client`) on nodes associated with an AWS OpsWorks for Chef Automate server is 13.*x*. We recommend running the most current, stable `chef-client` version, or at least 14.10.9.

Topics

- Compliance in Chef Automate 2.0
- Compliance in Chef Automate 1.x
- Updates to Compliance
- Community and Custom Compliance Profiles
- See Also

Compliance in Chef Automate 2.0

If your AWS OpsWorks for Chef Automate server is running Chef Automate 2.0, set up Chef Compliance by using procedures in this section.

Running Compliance Scan Jobs with Chef Automate 2.0

Chef Automate 2.0 includes Chef InSpec compliance-scanning capability that formerly required manual setup and cookbook configuration. You can run *scan jobs* on an AWS OpsWorks for Chef Automate server that is running Chef Automate 2.0. Jobs can be run immediately (one time), scheduled for a later time, or scheduled to run at specified intervals, such as daily or every two hours. The results of a scan job are sent to compliance reporting. You can view and take actions on compliance scan results in the Chef Automate dashboard. To open the **Compliance** tab and view reports, on the **Scan Jobs** tab in the Chef Automate dashboard, choose **Report** at the right of a managed node row.

To run scan jobs on managed nodes, you must have the following.

- At least one compliance profile installed in your namespace.
- At least one target node, either manually-added, or an EC2 instance added automatically.

In AWS OpsWorks for Chef Automate, scan jobs are supported on the following targets.

- Manually-added nodes
- `aws-ec2` instances
- AWS Regions

For detailed instructions about how to run scan jobs, see Chef Automate Scan Jobs in the Chef documentation.

(Optional, Chef Automate 2.0) Setting Up Compliance with the Audit Cookbook

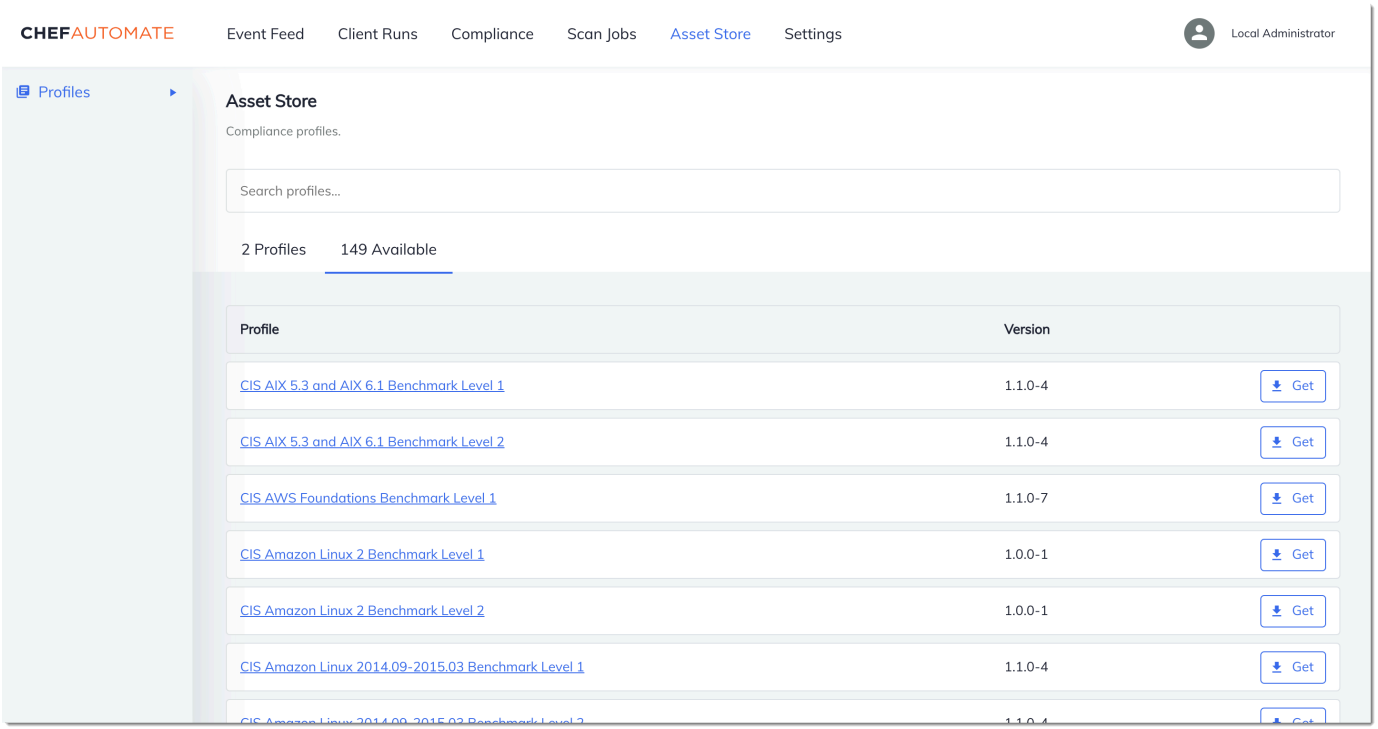
You can configure compliance on any AWS OpsWorks for Chef Automate server. After you launch an AWS OpsWorks for Chef Automate server, you can install profiles from the Chef Automate dashboard, or add desired profiles to Audit cookbook attributes in the `Policyfile.rb` policy file. A prefilled `Policyfile.rb` file is included in the starter kit.

After you edit `Policyfile.rb` with profiles as attributes of the Audit cookbook, run `chef push` commands to upload the Audit cookbook and other cookbooks specified in `Policyfile.rb` to your Chef Automate server. Installing the Audit cookbook also installs the gem for Chef InSpec, an open-source testing and auditing framework produced by Chef. For Chef Automate 2.0, choose version 7.1.0 or later of the Audit cookbook. The InSpec gem must be version 2.2.102 or later.

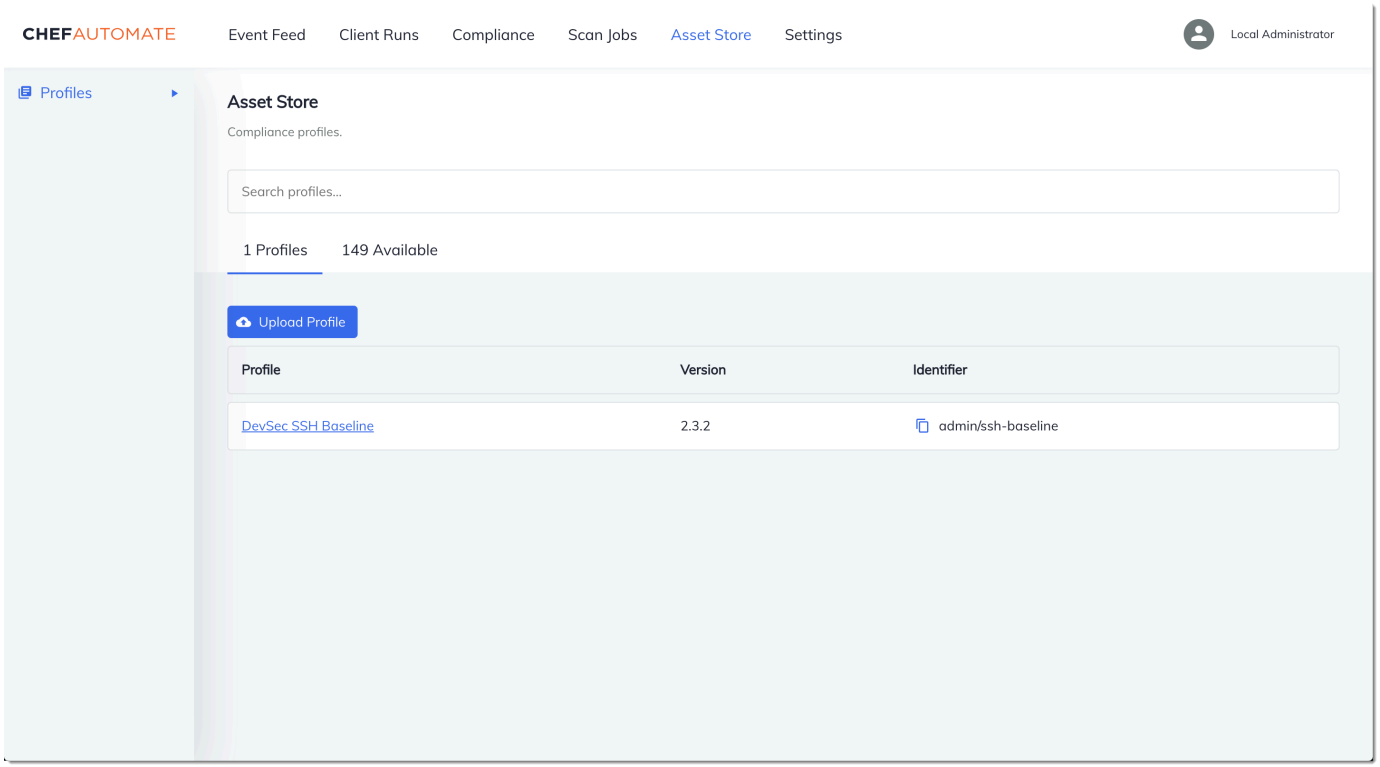
Instructions in this section show you how to implement the `opsworks-audit` cookbook. The Audit cookbook downloads specified profiles from the Chef Automate server, assesses nodes against the **DevSec SSH Baseline** profile, and reports the result of compliance scans on every `chef-client` run.

To install compliance profiles

1. If you have not already done so, sign in to the Chef Automate web-based dashboard. Use the credentials you received when you downloaded the Starter Kit as you created your AWS OpsWorks for Chef Automate server.
2. In the Chef Automate dashboard, choose the **Asset Store** tab.



3. Choose the **Available** tab to see predefined profiles.
4. Browse the list of profiles. Choose a profile that matches the operating system and configuration of at least one of your managed nodes. To view details about the profile, including a description of the profile's targeted violations and underlying rule code, choose **>** at the right of the profile entry. You can choose multiple profiles. If you are setting up the example in the Starter Kit, choose **DevSec SSH Baseline**.



5. To install the selected profiles on your Chef Automate server, choose **Get**.
6. After you install profiles, they are shown in the **Profiles** tab of the Chef Automate dashboard.

To install cookbooks with `Policyfile.rb`

1. View `Policyfile.rb` in your Starter Kit to see that the attributes for the Audit cookbook specify the `ssh-baseline` profile in `['profiles']`.

```
# Define audit cookbook attributes
default["opsworks-demo"]["audit"]["reporter"] = "chef-server-automate"
default["opsworks-demo"]["audit"]["profiles"] = [
  {
    "name": "DevSec SSH Baseline",
    "compliance": "admin/ssh-baseline"
  }
]
```

2. Download and install the cookbooks defined in `Policyfile.rb`.

```
chef install
```

All cookbooks are versioned in the cookbook's `metadata.rb` file. Each time you change a cookbook, you must raise the version of the cookbook that is in its `metadata.rb`.

3. Push the policy `opsworks-demo`, defined in `Policyfile.rb`, to your server.

```
chef push opsworks-demo
```

4. Verify the installation of your policy. Run the following command.

```
chef show-policy
```

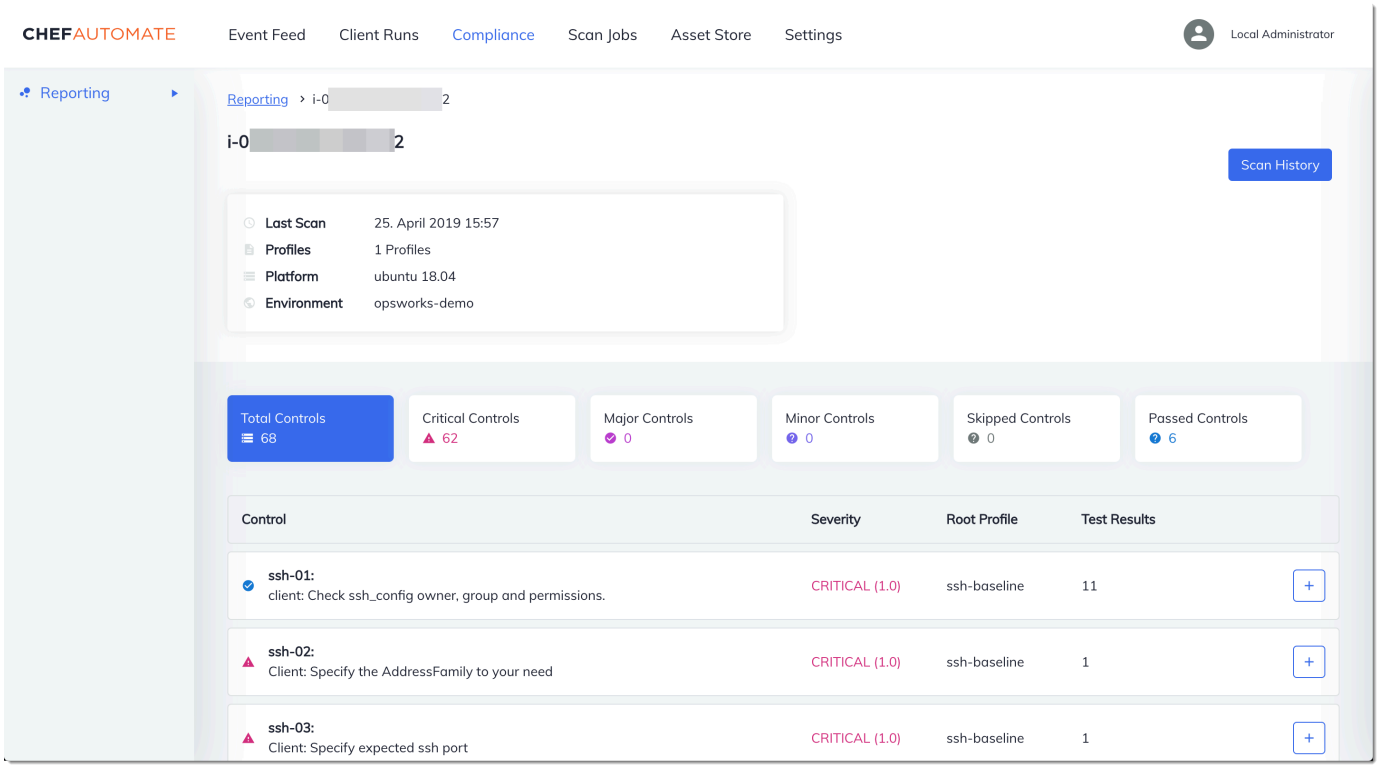
The results should resemble the following:

```
opsworks-demo-webserver
=======================
* opsworks-demo:  ec0fe46314
```

5. Add nodes to your server to manage, if you have not already done so. To connect your first node to the AWS OpsWorks for Chef Automate server, use the `userdata.sh` script that is included in this Starter Kit. It uses the OpsWorks `AssociateNode` API to connect a node to your server.

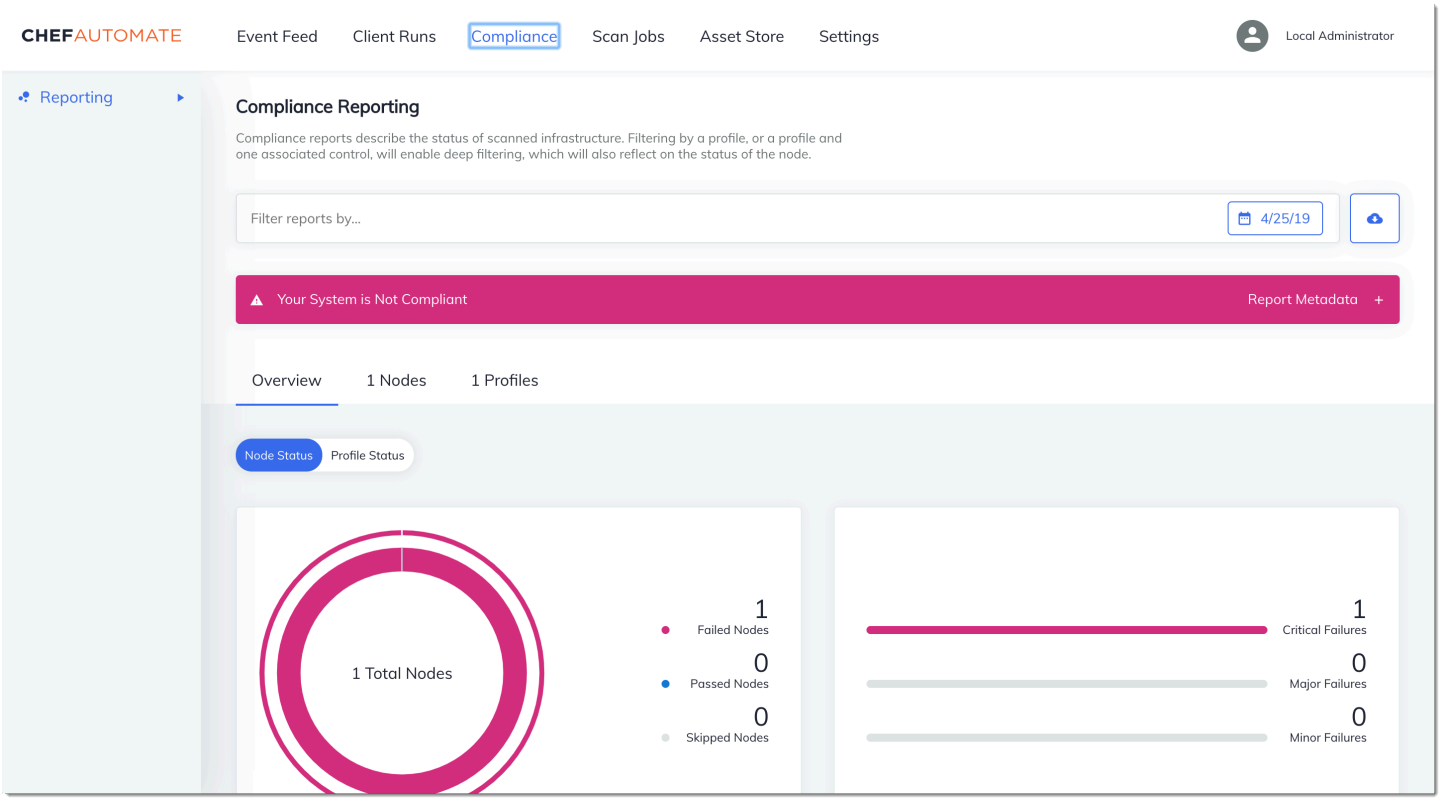
   You can automate the association of nodes by following steps in Add nodes automatically in AWS OpsWorks for Chef Automate, or add nodes one at a time by following steps in Add nodes individually.

6. After you update the run list for your nodes, the `chef-client` agent runs your specified recipes on its next run. By default, this occurs every 1800 seconds (30 minutes). After the run, you can view and take actions on compliance results from the **Compliance** tab in the Chef Automate dashboard.

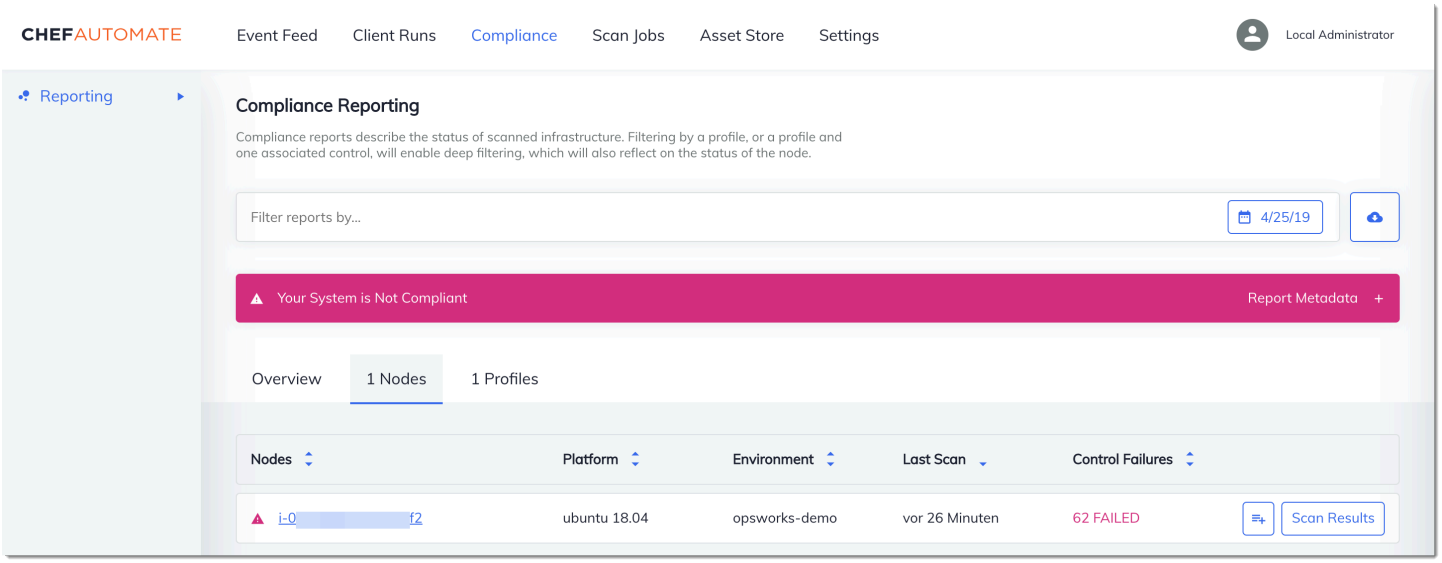


Running a Compliance Scan

You should see compliance scan results in the Chef Automate dashboard shortly after the first run of the agent that occurs after you configure node run lists.



In the Chef Automate dashboard, choose the **Compliance** tab. In the left navigation pane, choose **Reporting**. Choose the **Profiles** tab, choose **Scan Results**, and then choose a node with scan failures to learn more about the rules against which a node failed.



Typically, you see noncompliant scan results, because new nodes do not yet satisfy all rules in the **DevSec SSH Baseline** profile. The DevSec Hardening Framework, a community-based project, offers cookbooks to fix issues that violate the rules in the **DevSec SSH Baseline** profile.

(Optional) Resolving Noncompliant Results

The starter kit includes an open-source cookbook, `ssh-hardening`, that you can run to fix noncompliant results from runs against the **DevSec SSH Baseline** profile.

> **Note**
>
> The `ssh-hardening` cookbook makes changes to your nodes to comply with **DevSec SSH Baseline** rules. Before running this cookbook on any production nodes, review details about the **DevSec SSH Baseline** profile in the Chef Automate console to understand the rule violations that the cookbook targets. Review information about the open-source `ssh-hardening` cookbook before running it on any production nodes.

To run the `ssh-hardening` cookbook

1. In a text editor, append the `ssh-hardening` cookbook to the run list of `Policyfile.rb`. The `Policyfile.rb` run list should match the following.

   ```
   run_list  'chef-client', 'opsworks-webserver', 'audit', 'ssh-hardening'
   ```

2. Update `Policyfile.rb`, and push it to your AWS OpsWorks for Chef Automate server.

   ```
   chef update Policyfile.rb
      chef push opsworks-demo
   ```

3. Nodes that are associated with the `opsworks-demo` policy update the run list automatically, and apply the `ssh-hardening` cookbook on the next `chef-client` run.

   Because you are using the `chef-client` cookbook, your node checks in at regular intervals (by default, every 30 minutes). On the next check-in, the `ssh-hardening` cookbook runs, and helps improve node security to meet the **DevSec SSH Baseline** profile's rules.

4. After the initial run of the `ssh-hardening` cookbook, wait 30 minutes for a compliance scan to run again. View the results in the Chef Automate dashboard. The noncompliant results that occurred in the initial run of the **DevSec SSH Baseline** scan should be resolved.

Compliance in Chef Automate 1.*x*

If your AWS OpsWorks for Chef Automate server is running Chef Automate 1.*x*, set up Chef Compliance by using procedures in this section.

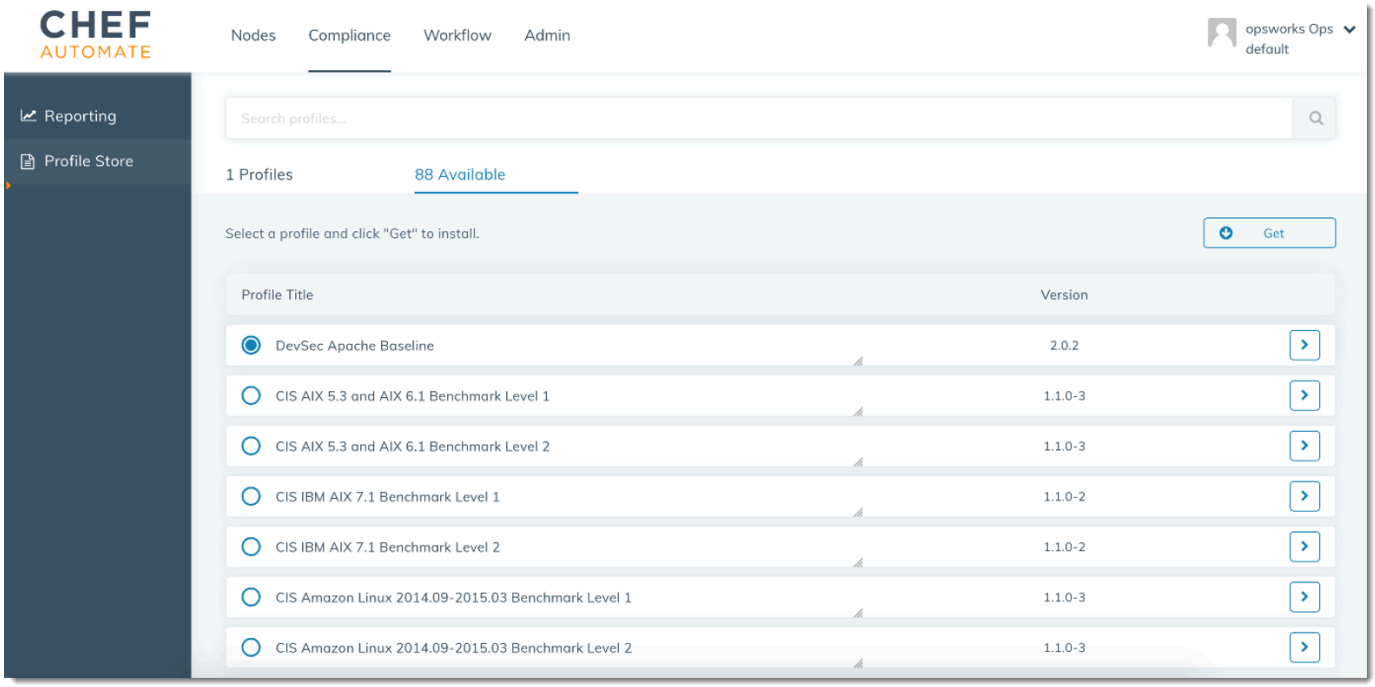
(Optional, Chef Automate 1.*x*) Setting Up Chef Compliance

You can configure Chef Compliance on any AWS OpsWorks for Chef Automate server. After you launch an AWS OpsWorks for Chef Automate server, choose profiles that you want to run from the profiles in the Chef Automate dashboard. After you install profiles, run `berks` commands to upload the Audit cookbook to your Chef Automate server. Installing the Audit cookbook also installs the gem for InSpec, an open-source testing framework produced by Chef that lets you integrate automated tests into any stage of your deployment pipeline. For Chef Automate 1.*x*, choose version 5.0.1 or later of the Audit cookbook. The InSpec gem must be version 1.24.0 or later.

The AWS OpsWorks for Chef Automate starter kit includes a wrapper cookbook, `opsworks-audit`, that downloads and installs the right version of Chef's Audit cookbook for you. The `opsworks-audit` cookbook also instructs the `chef-client` agent to assess nodes against the **DevSec SSH Baseline** profile that you install from Chef's Compliance console later in this topic. You can set up Compliance by using either cookbook to suit your preference. Instructions in this section show you how to implement the `opsworks-audit` cookbook.

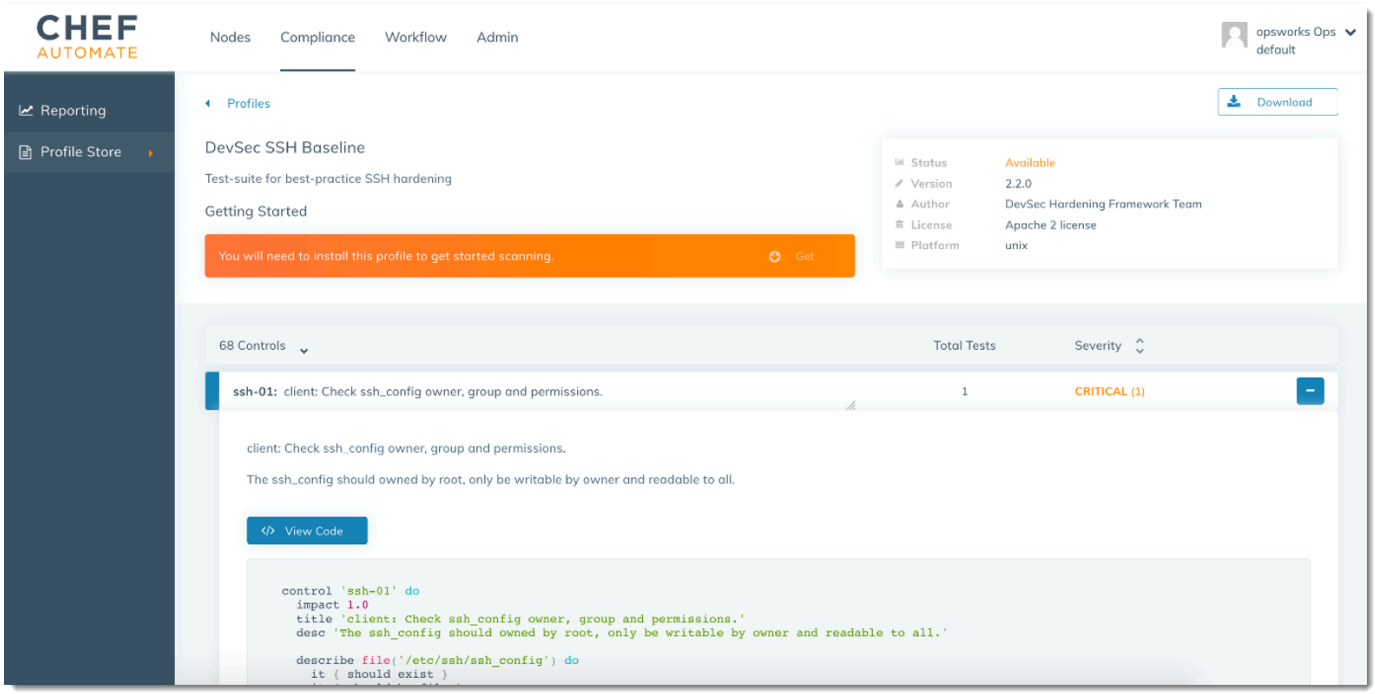
To install Compliance profiles

1. If you have not already done so, sign in to the Chef Automate web-based dashboard. Use the credentials you received when you downloaded the Starter Kit as you created your AWS OpsWorks for Chef Automate server.
2. In the Chef Automate dashboard, choose the **Compliance** tab.



3. In the left navigation bar, choose **Profile Store**, and then choose the **Available** tab to see predefined profiles.

4. Browse the list of profiles. Choose a profile that matches the operating system and configuration of at least one of your managed nodes. To view details about the profile, including a description of the profile's targeted violations and underlying rule code, choose **>** at the right of the profile entry. You can choose multiple profiles.



5. To install the selected profiles on your Chef Automate server, choose **Get**.
6. When the download is complete, go to the next procedure.

To install and set up the `opsworks-audit` cookbook

1. This step is optional, but it saves time in Step 6, when you are adding recipes to node run lists. Edit the `roles/opsworks-example-role.rb` file that is included in the starter kit you downloaded during the creation of your AWS OpsWorks for Chef Automate server. Add the following lines. The last line is commented out, because adding the `ssh-hardening` cookbook and recipe to resolve noncompliant nodes after your Compliance scan runs is optional.

   ```
   run_list(
        "recipe[chef-client]",
        "recipe[apache2]",
        "recipe[opsworks-audit]"
        # "recipe[ssh-hardening]"
          )
   ```

2. Use a text editor to specify your desired cookbooks in your Berksfile. A sample Berksfile is provided for you in the starter kit. In this example, we install the Chef Infra client (`chef-client`) cookbook, the `apache2` cookbook, and the `opsworks-audit` cookbook. Your Berksfile should resemble the following.

   ```
   source 'https://supermarket.chef.io
        cookbook 'chef-client'
        cookbook 'apache2', '~> 5.0.1'
        cookbook 'opsworks-audit', path: 'cookbooks/opsworks-audit', '~> 1.0.0'
   ```

   All cookbooks are versioned in the cookbook's `metadata.rb` file. Each time you change a cookbook, you must raise the version of the cookbook that is in its `metadata.rb`.

3. Run the following command to download and install the cookbooks to the `cookbooks` folder on your local or working computer.

   ```
   berks vendor cookbooks
   ```

4. Run the following command to upload the vendored cookbooks to your AWS OpsWorks for Chef Automate server.

```
knife upload .
```

5. Run the following command to verify the installation of the `opsworks-audit` cookbook by showing a list of cookbooks that are currently available on the server.

   ```
   knife cookbook list
   ```

6. Add nodes to your server to manage, if you have not already done so. You can automate the association of nodes by following steps in Add nodes automatically in AWS OpsWorks for Chef Automate, or add nodes one at a time by following steps in Add nodes individually. Edit the run list of your nodes to add the role you specified in Step 1, `opsworks-example-role`. In this example, we edit the `RUN_LIST` attribute in the `userdata` script that you use to automate the association of nodes.

   ```
   RUN_LIST="role[opsworks-example-role]"
   ```

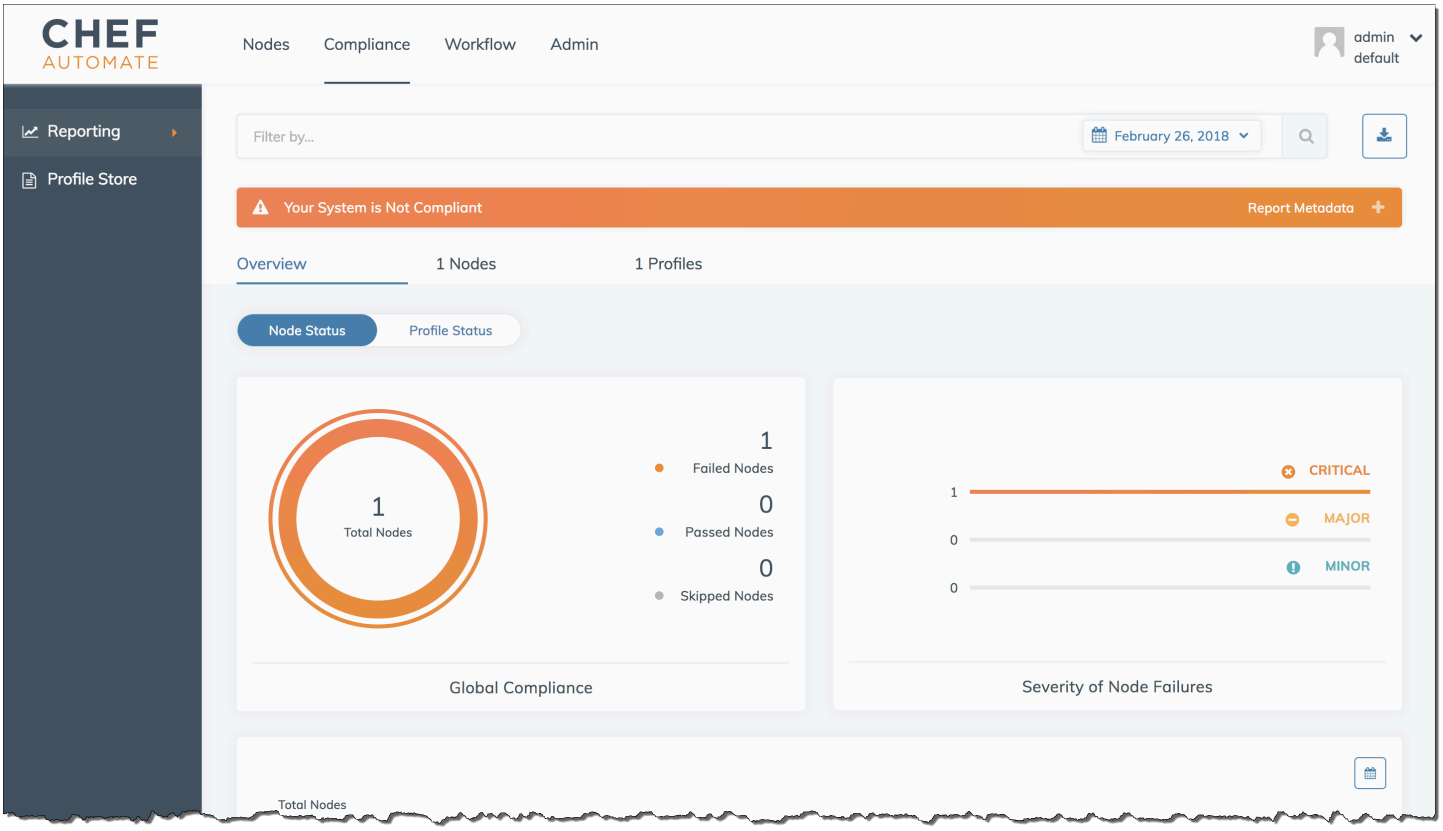
   If you skipped Step 1, and did not set up the role, add the names of the individual recipes to the run list. Save your changes, and follow steps in Step 3: Create Instances by Using an Unattended Association Script to apply your userdata script to Amazon EC2 instances.

   ```
   RUN_LIST="recipe[chef-client],recipe[apache2],recipe[opworks-audit]"
   ```

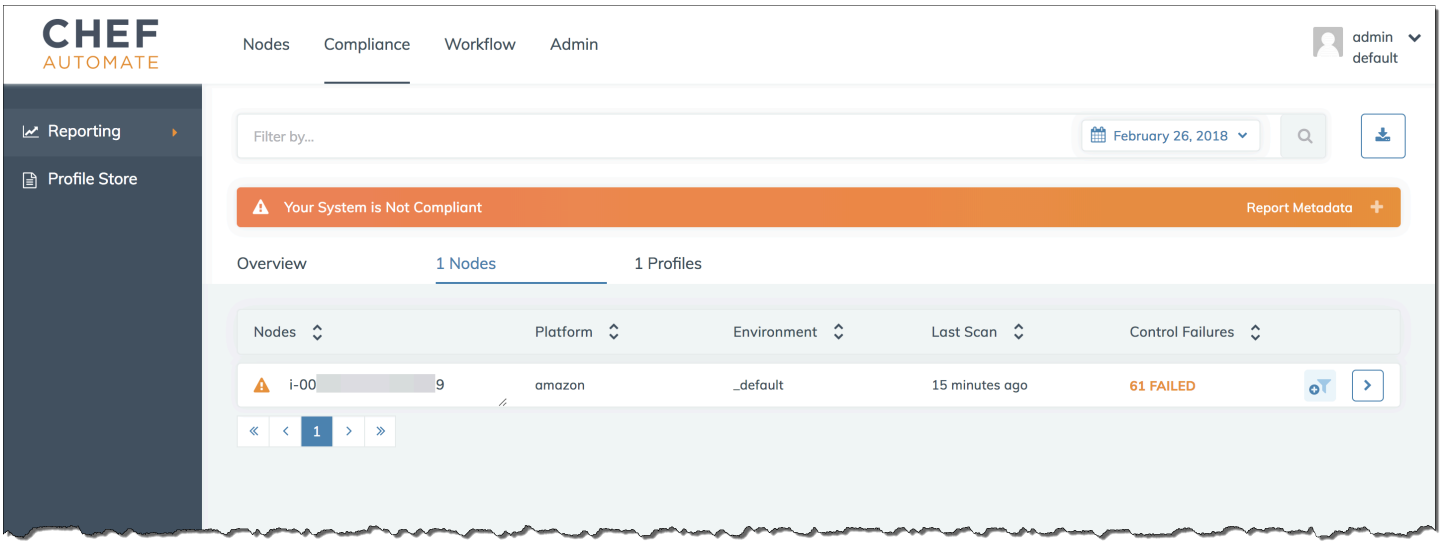
7. After you update the run list for your nodes, the `chef-client` agent runs your specified recipes on its next run. By default, this occurs every 1800 seconds (30 minutes). After the run, your Compliance results are visible in the Chef Automate dashboard.

Running a Compliance Scan

You should see compliance scan results in the Chef Automate dashboard shortly after the first run of the agent daemon that occurs after you configure node run lists.



In the Chef Automate dashboard, choose the **Compliance** tab. In the left navigation pane, choose **Reporting**. Choose the **Profiles** tab, choose **Scan Results**, and then choose a node with scan failures to learn more about the rules against which a node failed.



Typically, you see noncompliant scan results, because new nodes do not yet satisfy all rules in the **DevSec SSH Baseline** profile. The DevSec Hardening Framework, a community-based project, offers cookbooks to fix issues that violate the rules in the **DevSec SSH Baseline** profile.

(Optional) Resolving Noncompliant Results

The starter kit includes an open-source cookbook, `ssh-hardening`, that you can run to fix noncompliant results from runs against the **DevSec SSH Baseline** profile.

> **Note**
>
> The `ssh-hardening` cookbook makes changes to your nodes to comply with **DevSec SSH Baseline** rules. Before running this cookbook on any production nodes, review details about the **DevSec SSH Baseline** profile in the Chef Automate console to understand the rule violations that the cookbook targets. Review information about the open-source `ssh-hardening` cookbook before running it on any production nodes.

To run the `ssh-hardening` cookbook

1. In a text editor, append the `ssh-hardening` cookbook to your Berksfile. Your Berksfile should resemble the following.

   ```
   source 'https://supermarket.chef.io'
        cookbook 'chef-client'
        cookbook 'apache2', '~> 5.0.1'
        cookbook 'opsworks-audit', path: 'cookbooks/opsworks-audit', '~> 1.0.0' #
    optional
        cookbook 'ssh-hardening'
   ```

2. Run the following commands to download the `ssh-hardening` cookbook to your local cookbooks folder, and then upload it to your AWS OpsWorks for Chef Automate server.

   ```
   berks vendor cookbooks
   knife upload .
   ```

3. Add the `ssh-hardening` recipe to your node run list as described in Steps 1 and 6 of To install and set up the `opsworks-audit` cookbook.

   If you update the `opsworks-example-role.rb` file, upload your changes to your server by running the following command.

   ```
   knife upload .
   ```

If you update the run list directly, upload changes by running the following command. The node name is typically the instance ID.

```
knife node run_list add <node name> 'recipe[ssh-hardening]'
```

4. Because you are using the `chef-client` cookbook, your node checks in at regular intervals (by default, every 30 minutes). On the next check-in, the `ssh-hardening` cookbook runs, and helps improve node security to meet the **DevSec SSH Baseline** profile's rules.
5. After the initial run of the `ssh-hardening` cookbook, wait 30 minutes for a Compliance scan to run again. View the results in the Chef Automate dashboard. The noncompliant results that occurred in the initial run of the **DevSec SSH Baseline** scan should be resolved.

Updates to Compliance

On an AWS OpsWorks for Chef Automate server, compliance functionality is updated automatically by your scheduled system maintenance. As updated releases of Chef Automate, Chef Infra Server, and Chef InSpec become available for your AWS OpsWorks for Chef Automate server, you might need to check and update the supported versions of the Audit cookbook and Chef InSpec gem that are running on your server. Profiles that you have already installed on your AWS OpsWorks for Chef Automate server are not updated as part of maintenance.

Community and Custom Compliance Profiles

Chef currently includes over 100 compliance scan profiles. You can add community and custom profiles to the list, and then download and run compliance scans based on those profiles, just as you would for included profiles. Community-based compliance profiles are available from the Chef Supermarket. Custom profiles are Ruby-based programs that include a folder of controls that specify your scan rules.

See Also

- Chef Compliance announcement blog post
- Chef Automate Compliance online training
- Chef InSpec website
- Chef InSpec Tutorials

Disassociate a Node from an AWS OpsWorks for Chef Automate Server

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section describes how to disassociate, or remove, a managed node from management by an AWS OpsWorks for Chef Automate server. This operation is performed on the command line; you cannot disassociate nodes in the AWS OpsWorks for Chef Automate management console. Currently, the AWS OpsWorks for Chef Automate API does not allow for batch removal of multiple nodes. The command in this section disassociates one node at a time.

We recommend that you disassociate nodes from a Chef server before you delete the server, so that the nodes continue to operate without trying to reconnect with the server. To do this, run the `disassociate-node` AWS CLI command.

To disassociate nodes

1. In the AWS CLI, run the following command to disassociate nodes. *`Node_name`* is the name of the node that you want to disassociate; for Amazon EC2 instances, this is the instance ID. *`Server_name`* is the name of the Chef server from which you want to disassociate the node. `--engine-attributes` specifies your default `CHEF_AUTOMATE_ORGANIZATION` name. All three of these parameters are required.

   The `--region` parameter is not required unless you want to disassociate a node from a Chef server that is not in your default region.

   ```
   aws opsworks-cm --region Region_name disassociate-node --node-
   name Node_name --server-name Server_name --engine-attributes
    "Name=CHEF_AUTOMATE_ORGANIZATION,Value='default'"
   ```

   The following command is an example.

```
aws opsworks-cm --region us-west-2 disassociate-node --node-name
 i-0010zzz00d66zzz90 --server-name opsworkstest --engine-attributes
 "Name=CHEF_AUTOMATE_ORGANIZATION,Value='default'"
```

2. Wait until a response message indicates that the disassociation is finished.

   After you successfully disassociate a node from an AWS OpsWorks for Chef Automate server, it might still be visible in the Chef Automate dashboard. By default, Chef enforces a retention period for node state information, and purges the node automatically after a few days.

For more information about how to delete an AWS OpsWorks for Chef Automate server, see Delete an AWS OpsWorks for Chef Automate Server.

Related Topics

The following AWS blog posts offer more information about automatically associating nodes with your Chef Automate server, by using Auto Scaling groups, or within multiple accounts.

- Using AWS OpsWorks for Chef Automate to Manage EC2 Instances with Auto Scaling
- OpsWorks for Chef Automate – Automatically Bootstrapping Nodes in Different Accounts

Delete an AWS OpsWorks for Chef Automate Server

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section describes how to delete an AWS OpsWorks for Chef Automate server. Deleting a server also deletes its events, logs, and any cookbooks that were stored on the server. Supporting resources (Amazon Elastic Compute Cloud instance, Amazon Elastic Block Store volume, etc.) are deleted also, along with all automated backups.

Although deleting a server does not delete nodes, they are no longer managed by the deleted server, and will continuously attempt to reconnect. For this reason, we recommend disassociating managed nodes before you delete a Chef server. In this release, you can disassociate nodes by running an AWS CLI command.

Step 1: Disassociate Managed Nodes

Disassociate nodes from the Chef server before you delete the server, so that the nodes continue to operate without trying to reconnect with the server. To do this, run the `disassociate-node` AWS CLI command.

To disassociate nodes

1. In the AWS CLI, run the following command to disassociate nodes. *`Server_name`* is the name of the Chef server from which you want to disassociate the node.

   ```
   aws opsworks-cm --region Region_name disassociate-node --node-name Node_name --
   server-name Server_name
   ```

2. Wait until a response message indicates that the disassociation is finished.

Step 2: Delete the Server

1. On the server's tile on the dashboard, expand the **Actions** menu.
2. Choose **Delete server**.
3. When you are prompted to confirm the deletion, choose **Yes**.

Reset Chef Automate Dashboard Credentials

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Periodically, you might want to change the password with which you sign in to the Chef Automate dashboard. You can also use the Amazon EC2 Systems Manager AWS CLI commands shown in this section to change the Chef Automate dashboard password if you have lost it. The command you use depends on whether your Chef Automate server is running version 1 or version 2 of Chef Automate.

1. To return the instance ID of your Chef server, open the AWS Management Console to the following page.

   https://console.aws.amazon.com/ec2/v2/home?region=*region_of_your_server*#Instances:search=aws-opsworks-cm-*server_name*

   For example, for a Chef server named **MyChefServer** in the US West (Oregon) Region, the console URL would be the following.

   https://console.aws.amazon.com/ec2/v2/home?region=us-west-2#Instances:search=aws-opsworks-cm-MyChefServer

   Make a note of the instance ID that is displayed in the console; you will need it to change your password.

2. To reset the Chef Automate dashboard sign-in password, run one of the following AWS CLI commands, depending on whether your server is running Chef Automate 1 or Chef Automate 2. Replace *enterprise_name* with your enterprise or organization name, *user_name* with the user name of an administrator on the server, *new_password* with the password you want to use, and*region_name* with the region in which your server is located. If you do not specify an enterprise name, the enterprise name will be `default`. By default, *enterprise_name* is `default` (this is the name of the organization that is always provisioned). For *user_name*, AWS OpsWorks for Chef Automate only creates a user named `admin`. Make a note of the new password, and store it in a safe but convenient location.

   For Chef Automate 1:

   ```
   aws ssm send-command --document-name "AWS-RunShellScript" --comment "reset admin
    password" --instance-ids "instance_id"
   --parameters commands="sudo delivery-ctl reset-
   password enterprise_name user_name new_password" --region region_name --output text
   ```

   For Chef Automate 2:

```
aws ssm send-command --document-name "AWS-RunShellScript" --comment "reset admin
 password" --instance-ids "instance_id"
--parameters commands="sudo chef-automate iam admin-access restore new_password" --
region region_name --output text
```

3. Wait for output text (in this case, the command ID) to show that the password change is finished.

Logging AWS OpsWorks for Chef Automate API Calls with AWS CloudTrail

> **Important**
>
> AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

AWS OpsWorks for Chef Automate is integrated with AWS CloudTrail, a service that provides a record of actions taken by an IAM identity, or an AWS service in AWS OpsWorks for Chef Automate. CloudTrail captures all API calls for AWS OpsWorks for Chef Automate as events, including calls from the AWS OpsWorks for Chef Automate console and from code calls to the AWS OpsWorks for Chef Automate APIs. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS OpsWorks for Chef Automate. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to AWS OpsWorks for Chef Automate, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

AWS OpsWorks for Chef Automate Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS OpsWorks for Chef Automate, that activity is recorded in a CloudTrail event along with other

AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for AWS OpsWorks for Chef Automate, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all regions. The trail logs events from all regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All AWS OpsWorks for Chef Automate actions are logged by CloudTrail and are documented in the AWS OpsWorks for Chef Automate API Reference. For example, calls to the **CreateServer**, **CreateBackup**, and **DescribeServers** actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understanding AWS OpsWorks for Chef Automate Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following example shows a CloudTrail log entry for the AWS OpsWorks for Chef Automate `CreateServer` action.


```
{"eventVersion":"1.05",
"userIdentity":{
    "type":"AssumedRole",
    "principalId":"ID number:OpsWorksCMUser",
    "arn":"arn:aws:sts::831000000000:assumed-role/Admin/OpsWorksCMUser",
    "accountId":"831000000000","accessKeyId":"ID number",
    "sessionContext":{
        "attributes":{
            "mfaAuthenticated":"false",
            "creationDate":"2017-01-05T22:03:47Z"
            },
        "sessionIssuer":{
            "type":"Role",
            "principalId":"ID number",
            "arn":"arn:aws:iam::831000000000:role/Admin",
            "accountId":"831000000000",
            "userName":"Admin"
            }
        }
    },
"eventTime":"2017-01-05T22:18:23Z",
"eventSource":"opsworks-cm.amazonaws.com",
"eventName":"CreateServer",
"awsRegion":"us-west-2",
"sourceIPAddress":"101.25.190.51",
"userAgent":"console.amazonaws.com",
"requestParameters":{
    "serverName":"OpsChef-test-server",
    "engineModel":"Single",
    "engine":"Chef",
    "instanceProfileArn":"arn:aws:iam::831000000000:instance-profile/aws-opsworks-cm-
ec2-role",
    "backupRetentionCount":3,"serviceRoleArn":"arn:aws:iam::831000000000:role/service-
role/aws-opsworks-cm-service-role",
    "engineVersion":"12",
    "preferredMaintenanceWindow":"Fri:21:00",
    "instanceType":"t2.medium",
    "subnetIds":["subnet-1e111f11"],
    "preferredBackupWindow":"Wed:08:00"
    },
```

```
"responseElements":{
    "server":{
        "endpoint":"OpsChef-test-server-thohsgreckcnwgz3.us-west-2.opsworks-cm.io",
        "createdAt":"Jan 5, 2017 10:18:22 PM",
        "serviceRoleArn":"arn:aws:iam::831000000000:role/service-role/aws-opsworks-cm-
service-role",
        "preferredBackupWindow":"Wed:08:00",
        "status":"CREATING",
        "subnetIds":["subnet-1e111f11"],
        "engine":"Chef",
        "instanceType":"t2.medium",
        "serverName":"OpsChef-test-server",
        "serverArn":"arn:aws:opsworks-cm:us-west-2:831000000000:server/OpsChef-test-
server/8epp7f6z-e91f-4z10-89z5-8c6219cdb09f",
        "engineModel":"Single",
        "backupRetentionCount":3,
        "engineAttributes":[
            {"name":"CHEF_STARTER_KIT","value":"*** Redacted ***"},
            {"name":"CHEF_PIVOTAL_KEY","value":"*** Redacted ***"},
            {"name":"CHEF_DELIVERY_ADMIN_PASSWORD","value":"*** Redacted ***"}],
        "engineVersion":"12.11.1",
        "instanceProfileArn":"arn:aws:iam::831000000000:instance-profile/aws-opsworks-
cm-ec2-role",
        "preferredMaintenanceWindow":"Fri:21:00"
        }
    },
"requestID":"de7f64f9-d394-12ug-8081-7bb0386fbcb6",
"eventID":"8r7b18df-6c90-47be-87cf-e8346428cfc3",
"eventType":"AwsApiCall",
"recipientAccountId":"831000000000"
}
```


Troubleshooting AWS OpsWorks for Chef Automate

Important

AWS OpsWorks for Chef Automate reached end of life on May 5, 2024 and has been disabled for both new and existing customers. We recommend that existing customers migrate to Chef SaaS or an alternative solution. If you have questions, you can reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This topic contains some common AWS OpsWorks for Chef Automate issues, and suggested solutions for those issues.

Topics

- General troubleshooting tips
- Troubleshooting specific errors
- Additional help and support

General troubleshooting tips

If you are unable to create or work with a Chef server, you can view error messages or logs to help you troubleshoot the issue. The following tasks describe general places to start when you are troubleshooting a Chef server issue. For information about specific errors and solutions, see the Troubleshooting specific errors section of this topic.

- Use the AWS OpsWorks for Chef Automate console to view error messages if a Chef server fails to start. On the Chef server detail page, error messages related to launching and running the server are shown at the top of the page. Errors can come from AWS OpsWorks for Chef Automate, AWS CloudFormation, or Amazon EC2, services that are used to create a Chef server. On the detail page, you can also view events that occur on a running server, which can contain failure event messages.
- To help resolve EC2 issues, connect to your server's instance by using SSH, and view logs. EC2 instance logs are stored in the `/var/log/aws/opsworks-cm` directory. These logs capture command outputs while AWS OpsWorks for Chef Automate launches a Chef server.

Troubleshooting specific errors

Topics

- Server is in a Connection lost state
- Managed node shows up in the Chef Automate dashboard in the Missing column
- Cannot create a Chef vault; knife vault command fails with errors
- Server creation fails with "requested configuration is currently not supported" message
- Chef server doesn't recognize organization names added in the Chef Automate dashboard
- Unable to create the server's Amazon EC2 instance

- Service role error prevents server creation
- Elastic IP address limit exceeded
- Cannot sign into the Chef Automate dashboard
- Unattended node association fails
- System maintenance fails

Server is in a Connection lost state

Problem: A server's status shows as **Connection lost**.

Cause: This most commonly occurs when an entity outside of OpsWorks makes changes to an AWS OpsWorks for Chef Automate server or its supporting resources. OpsWorks cannot connect to Chef Automate servers in **Connection lost** states to handle maintenance tasks such as creating backups, applying operating system patches, or updating Chef Automate. As a result, your server might be missing important updates, susceptible to security issues, or otherwise not operating as expected.

Solution: Try the following steps to restore the server's connection.

1. Be sure that your service role has all required permissions.

    a. On the **Settings** page for your server, in **Network and security**, choose the link for the service role that the server is using. This opens the service role for viewing in the IAM console.

    b. On the **Permissions** tab, verify that `AWSOpsWorksCMServiceRole` is in the **Permissions policies** list. If it isn't listed, add the `AWSOpsWorksCMServiceRole` managed policy manually to the role.

    c. On the **Trust relationships** tab, verify that the service role has a trust policy that trusts the `opsworks-cm.amazonaws.com` service to assume roles on your behalf. For more information about how to use trust policies with roles, see Modifying a role (console), or the AWS Security Blog post, How to use trust policies with IAM roles.

2. Be sure that your instance profile has all required permissions.

    a. On the **Settings** page for your server, in **Network and security**, choose the link for the instance profile that the server is using. This opens the instance profile for viewing in the IAM console.

   b. On the **Permissions** tab, verify that `AmazonEC2RoleforSSM` and `AWSOpsWorksCMInstanceProfileRole` are both in the **Permissions policies** list. If one or both aren't listed, add these managed policies manually to the role.

   c. On the **Trust relationships** tab, verify that the service role has a trust policy that trusts the `ec2.amazonaws.com` service to assume roles on your behalf. For more information about how to use trust policies with roles, see Modifying a role (console), or the AWS Security Blog post, How to use trust policies with IAM roles.

3. In the Amazon EC2 console, be sure that you are in the same region as the region of the AWS OpsWorks for Chef Automate server, and then restart the EC2 instance that your server is using.

   a. Choose the EC2 instance that is named `aws-opsworks-cm-instance-`*`server-name`*.

   b. On the **Instance state** menu, choose **Reboot instance**.

   c. Allow up to 15 minutes for your server to restart and be fully online.

4. In the AWS OpsWorks for Chef Automate console, on the server details page, verify that the server status is now **healthy**.

If the server status is still **Connection lost** after performing the preceding steps, try one of the following.

- Replace the server by creating a new one and deleting the original. If data on the current server is important to you, restore the server from a recent backup, and verify the data is up to date before deleting the original, unresponsive server.
- Contact AWS support.

Managed node shows up in the Chef Automate dashboard in the Missing column

Problem: A managed node is showing up in the Chef Automate dashboard's **Missing** column.

Cause: When a node doesn't connect to the Chef Automate server for more than 12 hours, and `chef-client` cannot run on the node, the node changes from the state it was in before the 12-hour period, and moves to the **Missing** column of the Chef Automate dashboard.

Solution: Verify that the node is online. Try running `knife node show` *`node_name`* `--run-list` to see whether `chef-client` is able to run on the node, or `knife node show -l`

node_name to display all information about the node. The node might be offline or disconnected from the network.

Cannot create a Chef vault; `knife vault` command fails with errors

Problem: You are trying to create a vault on your Chef Automate server (such as a vault for storing credentials for domain-joining Windows-based nodes) by running the `knife vault` command. The command returns an error message similar to the following.

```
WARN: Auto inflation of JSON data is deprecated. Please pass in the class to inflate or
 use #edit_hash (CHEF-1)
at /opt/chefdk/embedded/lib/ruby/2.3.0/forwardable.rb:189:in `edit_data'.Please see
 https://docs.chef.io/deprecations_json_auto_inflate.html
for further details and information on how to correct this problem.
WARNING: pivotal not found in users, trying clients.
ERROR: ChefVault::Exceptions::AdminNotFound: FATAL: Could not find pivotal in users or
 clients!
```

The pivotal user is not returned when you run `knife user list` remotely, but you can see the pivotal user in results when you run the `chef-server-ctl user-show` command locally on your Chef Automate server. In other words, your `knife vault` command cannot find the pivotal user, but you know it exists.

Cause: Though the pivotal user is considered the superuser in Chef, and has full permissions, it is not a member of any organization, including the `default` organization that is used in AWS OpsWorks for Chef Automate. The command `knife user list` returns all the users that are in the current organization in your Chef configuration. The `chef-server-ctl user-show` command returns all users regardless of organization, including the pivotal user.

Solution: To fix the problem, add the pivotal user to the default organization by running `knife opc`.

First, you'll need to install the knife-opc plugin.

```
chef gem install knife-opc
```

After you install the plugin, run the following command to add the pivotal user to the default organization.

```
knife opc org user add default pivotal
```

You can verify that the pivotal user is part of the default organization by running `knife user list` again. `pivotal` should be listed in the results. Then, try running `knife vault` again.

Server creation fails with "requested configuration is currently not supported" message

Problem: You are trying to create a Chef Automate server, but server creation fails with an error message that is similar to "The requested configuration is currently not supported. Please check the documentation for supported configurations."

Cause: An unsupported instance type might have been specified for the Chef Automate server. If you choose to create the Chef Automate server in a VPC that has a non-default tenancy, such as one for dedicated instances, all instances inside the specified VPC must also be of dedicated or host tenancy. Because some instance types, such as t2, are available only with default tenancy, the Chef Automate server instance type might not be supportable by the specified VPC, and server creation fails.

Solution: If you choose a VPC that has a non-default tenancy, use an m4 instance type, which can support dedicated tenancy.

Chef server doesn't recognize organization names added in the Chef Automate dashboard

Problem: You've added new Workflow organization names in the Chef Automate dashboard, or specified a `CHEF_AUTOMATE_ORGANIZATION` value other than `"default"` in the unattended node association script, but node association fails. Your AWS OpsWorks for Chef Automate server does not recognize the new organization names.

Cause: Workflow organization names and Chef server organization names are not the same. You can create new Workflow organizations in the web-based Chef Automate dashboard, but not Chef server organization names. You can use the Chef Automate dashboard only to view existing Chef server organizations. A new organization that you create in the Chef Automate dashboard is a Workflow organization, and is not recognized by the Chef server. You cannot create new organization names by specifying them in the node association script. Referring to an organization name in a node association script when the organization has not first been added to the Chef server will cause node association to fail.

Solution: To create new organizations that are recognized on the Chef server, use the `knife opc org create` command, or run `chef-server-ctl org-create`.

Unable to create the server's Amazon EC2 instance

Problem: Server creation failed with an error message similar to the following: "The following resource(s) failed to create: [EC2Instance]. Failed to receive 1 resource signal(s) within the specified duration."

Cause: This is most likely because the EC2 instance doesn't have network access.

Solution: Ensure the instance has outbound Internet access, and the AWS service agent is able to issue commands. Be sure that your VPC (a VPC with a single public subnet) has **DNS resolution** enabled, and that your subnet has the **Auto-assign Public IP** setting enabled.

Service role error prevents server creation

Problem: Server creation fails with an error message that states, "Not authorized to perform sts:AssumeRole."

Cause: This can occur when the service role you are using lacks adequate permissions to create a new server.

Solution: Open the AWS OpsWorks for Chef Automate console; use the console to generate a new service role and an instance profile role. If you would prefer to use your own service role, attach the **AWSOpsWorksCMServiceRole** policy to the role. Verify that **opsworks-cm.amazonaws.com** is listed among services in the role's **Trust relationships**. Verify that the service role that is associated with the Chef server has the **AWSOpsWorksCMServiceRole** managed policy attached.

Elastic IP address limit exceeded

Problem: Server creation fails with an error message that states, "The following resource(s) failed to create: [EIP, EC2Instance]. Resource creation cancelled, the maximum number of addresses has been reached."

Cause: This occurs when your account has used the maximum number of Elastic IP (EIP) addresses. The default EIP address limit is five.

Solution: You can either release existing EIP addresses or delete ones that your account is not actively using, or you can contact AWS Customer Support to increase the limit of EIP addresses that is associated with your account.

Cannot sign into the Chef Automate dashboard

Problem: The Chef Automate dashboard shows an error similar to the following: "Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at https://myserver-name.region.opsworks-cm.io/api/v0/e/default/verify-token. (Reason: CORS header 'Access-Control-Allow-Origin' missing)". The error can also be similar to "The User Id / Password combination entered is incorrect."

Cause: The Chef Automate dashboard explicity sets the FQDN, and does not accept relative URLs. At this time, you cannot sign in by using the Chef server's IP address; you can only sign in by using the DNS name of the server.

Solution: Sign in to the Chef Automate dashboard only by using the Chef server's DNS name entry, not its IP address. You can also try resetting the Chef Automate dashboard credentials by running an AWS CLI command, as described in Reset Chef Automate Dashboard Credentials.

Unattended node association fails

Problem: Unattended, or automatic, association of new Amazon EC2 nodes is failing. Nodes that should have been added to the Chef server are not showing up in the Chef Automate dashboard, and are not listed in results of the `knife client show` or `knife node show` commands.

Cause: This can occur when you do not have an IAM role set up as an instance profile that permits `opsworks-cm` API calls to communicate with new EC2 instances.

Solution: Attach a policy to your EC2 instance profile that allows the `AssociateNode` and `DescribeNodeAssociationStatus` API calls to work with EC2, as described in Add nodes automatically in AWS OpsWorks for Chef Automate.

System maintenance fails

AWS OpsWorks CM performs weekly system maintenance to ensure that the latest minor versions of Chef Server and Chef Automate Server, including security updates, are always running on an AWS OpsWorks for Chef Automate server. If, for any reason, system maintenance fails, AWS OpsWorks CM notifies you of the failure. For more information about system maintenance, see System Maintenance in AWS OpsWorks for Chef Automate.

This section describes possible reasons for failure and suggests solutions.

Topics

- Service role or instance profile error prevents system maintenance

Service role or instance profile error prevents system maintenance

Problem: System maintenance fails with an error message that states, "Not authorized to perform sts:AssumeRole", or a similar error message about permissions.

Cause: This can occur when either the service role or instance profile you are using lacks adequate permissions to perform system maintenance on the server.

Solution: Be sure that your service role and instance profile have all required permissions.

1. Be sure that your service role has all required permissions.

    a. On the **Settings** page for your server, in **Network and security**, choose the link for the service role that the server is using. This opens the service role for viewing in the IAM console.

    b. On the **Permissions** tab, verify that `AWSOpsWorksCMServiceRole` is attached to the service role. If `AWSOpsWorksCMServiceRole` is not listed, add this policy to the role.

    c. Verify that **opsworks-cm.amazonaws.com** is listed among services in the role's **Trust relationships**. For more information about how to use trust policies with roles, see Modifying a role (console), or the AWS Security Blog post, How to use trust policies with IAM roles.

2. Be sure that your instance profile has all required permissions.

    a. On the **Settings** page for your server, in **Network and security**, choose the link for the instance profile that the server is using. This opens the instance profile for viewing in the IAM console.

    b. On the **Permissions** tab, verify that `AmazonEC2RoleforSSM` and `AWSOpsWorksCMInstanceProfileRole` are both in the **Permissions policies** list. If one or both aren't listed, add these managed policies manually to the role.

    c. On the **Trust relationships** tab, verify that the service role has a trust policy that trusts the `ec2.amazonaws.com` service to assume roles on your behalf. For more information about how to use trust policies with roles, see Modifying a role (console), or the AWS Security Blog post, How to use trust policies with IAM roles.

Additional help and support

If you do not see your specific problem described in this topic, or you have tried the suggestions in this topic and are still having problems, visit the OpsWorks forums.

You can also visit the AWS Support Center. The AWS Support Center is the hub for creating and managing AWS Support cases. The AWS Support Center also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.

Security in AWS OpsWorks Configuration Management (CM)

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs 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 OpsWorks CM, 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.

This documentation helps you understand how to apply the shared responsibility model when using OpsWorks CM. The following topics show you how to configure OpsWorks CM to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your OpsWorks CM resources.

Topics

- Data Protection in OpsWorks CM
- Data Encryption
- Identity and Access Management for OpsWorks CM
- Internetwork Traffic Privacy
- Logging and Monitoring in OpsWorks CM
- Compliance Validation for OpsWorks CM
- Resilience in OpsWorks CM
- Infrastructure Security in AWS OpsWorks CM
- Configuration and Vulnerability Analysis in OpsWorks CM

- Security Best Practices for OpsWorks CM

Data Protection in OpsWorks CM

The AWS shared responsibility model applies to data protection in AWS OpsWorks Configuration Management. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with OpsWorks CM or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

The names of OpsWorks CM servers are not encrypted.

OpsWorks CM collects the following customer data in the course of creating and maintaining your AWS OpsWorks for Chef Automate and AWS OpsWorks for Puppet Enterprise servers.

- For OpsWorks for Puppet Enterprise, we collect private keys that Puppet Enterprise uses to enable communication between your Puppet master and managed nodes.
- For AWS OpsWorks for Chef Automate, we collect private keys for certificates that you attach to the service if you are using a custom domain. The private key that you provide when you are creating a Chef Automate server with a custom domain is passed through to your server.

OpsWorks CM servers store your configuration code, such as Chef cookbooks or Puppet Enterprise modules. Though this code is stored in server backups, AWS does not have access to it. This content is encrypted, and only administrators in your AWS account can access it. We recommend that you secure your Chef or Puppet configuration code using recommended protocols for your source repositories. For example, you can restrict permissions to repositories in AWS CodeCommit, or follow guidelines on the GitHub website for securing GitHub repositories.

OpsWorks CM does not use customer-provided content to maintain the service, or keep customer logs. Logs about your OpsWorks CM servers are stored in your account, in Amazon S3 buckets. IP addresses of users who connect to your OpsWorks CM servers are logged by AWS.

Integration with AWS Secrets Manager

Starting May 3, 2021, when you create a new server in OpsWorks CM, OpsWorks CM stores secrets for the server in AWS Secrets Manager. For new servers, the following attributes are stored as secrets in Secrets Manager.

- **Chef Automate server**
  - HTTPS private key (only servers that do not use a custom domain)
  - Chef Automate administrative password (CHEF_AUTOMATE_ADMIN_PASSWORD)
- **Puppet Enterprise master**
  - HTTPS private key (only servers that do not use a custom domain)
  - Puppet administrative password (PUPPET_ADMIN_PASSWORD)
  - Puppet r10k remote (PUPPET_R10K_REMOTE)

For existing servers that do not use a custom domain, the only secret stored in Secrets Manager, for both Chef Automate and Puppet Enterprise servers, is the HTTPS private key, because this is generated during automatic, weekly system maintenance.

OpsWorks CM stores secrets in Secrets Manager automatically, and this behavior is not user-configurable.

Data Encryption

OpsWorks CM encrypts server backups and communication between authorized AWS users and their OpsWorks CM servers. However, the root Amazon EBS volumes of OpsWorks CM servers are not encrypted.

Encryption at Rest

OpsWorks CM server backups are encrypted. However, the root Amazon EBS volumes of OpsWorks CM servers are not encrypted. This is not user-configurable.

Encryption in Transit

OpsWorks CM uses HTTP with TLS encryption. OpsWorks CM defaults to self-signed certificates to provision and manage servers, if no signed certificate is provided by users. We recommend that you use a certificate signed by a certificate authority (CA).

Key Management

AWS Key Management Service customer managed keys and AWS managed keys are not currently supported by OpsWorks CM.

Identity and Access Management for OpsWorks CM

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use OpsWorks CM resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating With Identities

- Managing Access Using Policies
- How OpsWorks CM Works with IAM
- AWS OpsWorks CM Identity-Based Policy Examples
- Troubleshooting AWS OpsWorks CM Identity and Access
- AWS managed policies for AWS OpsWorks Configuration Management
- Cross-service confused deputy prevention in AWS OpsWorks CM

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in OpsWorks CM.

Service user – If you use the OpsWorks CM service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more OpsWorks CM features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in OpsWorks CM, see Troubleshooting AWS OpsWorks CM Identity and Access.

Service administrator – If you're in charge of OpsWorks CM resources at your company, you probably have full access to OpsWorks CM. It's your job to determine which OpsWorks CM features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with OpsWorks CM, see How OpsWorks CM Works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to OpsWorks CM. To view example OpsWorks CM identity-based policies that you can use in IAM, see AWS OpsWorks CM Identity-Based Policy Examples.

Authenticating With Identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities.

When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and AWS Multi-factor authentication in IAM in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

IAM Users and Groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier

to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see Use cases for IAM users in the *IAM User Guide*.

> **Warning**
>
> IAM users have long-term credentials, which presents a security risk. To help mitigate this risk, we recommend that you provide these users with only the permissions they require to perform the task and that you remove these users when they are no longer needed.

IAM Roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. To temporarily assume an IAM role in the AWS Management Console, you can switch from a user to an IAM role (console). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Methods to assume a role in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Create a role for a third-party identity provider (federation) in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource

(instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Use an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Managing Access Using Policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their

permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-Based Policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choose between managed policies and inline policies in the *IAM User Guide*.

OpsWorks CM supports custom policies that you create in IAM and attach to users, roles, or groups.

Resource-Based Policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified

principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

OpsWorks CM does not support resource-based policies.

Access Control Lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

OpsWorks CM does not use ACLs.

Other Policy Types

OpsWorks CM does not support the following other policy types.

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to

any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs Work in the *AWS Organizations User Guide*.

- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple Policy Types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How OpsWorks CM Works with IAM

Before you use IAM to manage access to OpsWorks CM, you should understand what IAM features are available to use with OpsWorks CM. To get a high-level view of how OpsWorks CM and other AWS services work with IAM, see AWS services that work with IAM in the *IAM User Guide*.

Topics

- OpsWorks CM Identity-Based Policies
- OpsWorks CM and Resource-Based Policies
- Authorization Based on OpsWorks CM Tags
- OpsWorks CM IAM Roles

OpsWorks CM Identity-Based Policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. OpsWorks CM supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

In OpsWorks CM, you can attach a custom policy statement to a user, role, or group.

Actions

The `Action` element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in OpsWorks CM use the following prefix before the action: `opsworks-cm:`. For example, to grant someone permission to create an OpsWorks CM server by using an API operation, you include the `opsworks-cm:CreateServer` action in their policy. Policy statements must include either an `Action` or `NotAction` element. OpsWorks CM defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [
      "opsworks-cm:action1",
      "opsworks-cm:action2"
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "opsworks-cm:Describe*"
```

When you use wildcards to allow multiple actions in a policy statement, be careful that you are allowing those actions only for authorized services or users.

To see a list of OpsWorks CM actions, see Actions, Resources, and Condition Keys for AWS OpsWorks in the *IAM User Guide*.

Resources

The `Resource` element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. You specify a resource using an ARN or using the wildcard (*) to indicate that the statement applies to all resources.

You can get the Amazon Resource Number (ARN) of an OpsWorks CM server or backup by running the **DescribeServers** or **DescribeBackups** API operations, and base resource-level policies on those resources.

An OpsWorks CM server resource has an ARN in the following format:

```
arn:aws:opsworks-cm:{Region}:${Account}:server/${ServerName}/${UniqueId}
```

An OpsWorks CM backup resource has an ARN in the following format:

```
arn:aws:opsworks-cm:{Region}:${Account}:backup/${ServerName}-{Date-and-Time-Stamp-of-Backup}
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

For example, to specify the `test-chef-automate` Chef Automate server in your statement, use the following ARN:

```
"Resource": "arn:aws:opsworks-cm:us-west-2:123456789012:server/test-chef-automate/EXAMPLE-d1a2bEXAMPLE"
```

To specify all OpsWorks CM servers that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:opsworks-cm:us-west-2:123456789012:server/*"
```

The following example specifies an OpsWorks CM server backup as a resource:

```
"Resource": "arn:aws:opsworks-cm:us-west-2:123456789012:backup/test-chef-automate-server-2018-05-20T19:06:12.399Z"
```

Some OpsWorks CM actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

Many API actions involve multiple resources. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
      "resource1",
      "resource2"
```

To see a list of OpsWorks CM resource types and their ARNs, see Actions, Resources, and Condition Keys for AWS OpsWorks CM in the *IAM User Guide*. To learn with which actions you can specify the ARN of each resource, see Actions, Resources, and Condition Keys for AWS OpsWorks CM in the *IAM User Guide*.

Condition Keys

OpsWorks CM does not have service-specific context keys that can be used in the `Condition` element of policy statements. For the list of the global context keys that are available to all services, see AWS global condition context keys in the *IAM Policy Reference*. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can build conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant a user permission to access a resource only if it is tagged with the user's name. For more information, see IAM policy elements: Variables and tags in the *IAM User Guide*.

Examples

To view examples of OpsWorks CM identity-based policies, see AWS OpsWorks CM Identity-Based Policy Examples.

OpsWorks CM and Resource-Based Policies

OpsWorks CM does not support resource-based policies.

Resource-based policies are JSON policy documents that specify what actions a specified principal can perform on a resource and under what conditions.

Authorization Based on OpsWorks CM Tags

You can attach tags to OpsWorks CM resources or pass tags in a request to OpsWorks CM. To control access based on tags, you provide tag information in the condition element of a policy

using the `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys. For more information about tagging OpsWorks CM resources, see Working with Tags on AWS OpsWorks for Chef Automate Resources or Working with Tags on AWS OpsWorks for Puppet Enterprise Resources in this guide.

OpsWorks CM IAM Roles

An IAM role is an entity within your AWS account that has specific permissions.

OpsWorks CM uses two roles:

- A service role that grants the OpsWorks CM service permissions to work within a user's AWS account. If you use the default service role provided by OpsWorks CM, the name of this role is `aws-opsworks-cm-service-role`.
- An instance profile role that lets the OpsWorks CM service call the OpsWorks CM API. This role grants access to Amazon S3 and AWS CloudFormation to create the server and the S3 bucket for backups. If you use the default instance profile provided by OpsWorks CM, the name of this instance profile role is `aws-opsworks-cm-ec2-role`.

OpsWorks CM does not use service-linked roles.

Using Temporary Credentials with OpsWorks CM

OpsWorks CM supports using temporary credentials, and inherits that capability from AWS Security Token Service.

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

Service-Linked Roles

OpsWorks CM does not use service-linked roles.

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

Service Roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

OpsWorks CM uses two roles:

- A service role that grants the OpsWorks CM service permissions to work within a user's AWS account. If you use the default service role provided by OpsWorks CM, the name of this role is `aws-opsworks-cm-service-role`.
- An instance profile role that lets the OpsWorks CM service call the OpsWorks CM API. This role grants access to Amazon S3 and AWS CloudFormation to create the server and the S3 bucket for backups. If you use the default instance profile provided by OpsWorks CM, the name of this instance profile role is `aws-opsworks-cm-ec2-role`.

Choosing an IAM Role in OpsWorks CM

When you create a server in OpsWorks CM, you must choose a role to allow OpsWorks CM to access Amazon EC2 on your behalf. If you have already created a service role, then OpsWorks CM provides you with a list of roles to choose from. OpsWorks CM can create the role for you, if you do not specify one. It's important to choose a role that allows access to start and stop Amazon EC2 instances. For more information, see Create a Chef Automate Server or Create a Puppet Enterprise Master.

AWS OpsWorks CM Identity-Based Policy Examples

By default, users or roles don't have permission to create or modify OpsWorks CM resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant IAM identities permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating IAM policies in the *IAM User Guide*.

In OpsWorks CM, you can assign the `AWSOpsWorksCMServiceRole` policy to a user to let the user create and manage Chef Automate or Puppet Enterprise servers using either the AWS Management Console or AWS CLI.

Topics

- Policy Best Practices
- Allow Users to View Their Own Permissions
- Viewing OpsWorks CM Servers Based on Tags

Policy Best Practices

Identity-based policies determine whether someone can create, access, or delete OpsWorks CM resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and

functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.

- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Allow Users to View Their Own Permissions

This example shows how you might create a policy that allows users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

JSON

```
{
       "Version": "2012-10-17",
       "Statement": [
           {
               "Sid": "ViewOwnUserInfo",
               "Effect": "Allow",
               "Action": [
                   "iam:GetUserPolicy",
                   "iam:ListGroupsForUser",
                   "iam:ListAttachedUserPolicies",
                   "iam:ListUserPolicies",
                   "iam:GetUser"
               ],
               "Resource": [
                   "arn:aws:iam::*:user/${aws:username}"
               ]
           },
           {
               "Sid": "NavigateInConsole",
               "Effect": "Allow",
               "Action": [
                   "iam:GetGroupPolicy",
```

```
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Viewing OpsWorks CM Servers Based on Tags

You can use conditions in your identity-based policy to control access to OpsWorks CM servers and backups based on tags. This example shows how you might create a policy that allows viewing a OpsWorks CM server. However, permission is granted only if the OpsWorks CM server tag `Owner` has the value of that user's user name. This policy also grants the permissions necessary to complete this action on the console.

You can attach this policy to the users in your account. If a user named `richard-roe` attempts to view an OpsWorks CM server, the server must be tagged `Owner=richard-roe` or `owner=richard-roe`. Otherwise he is denied access. The condition tag key `Owner` matches both `Owner` and `owner` because condition key names are not case-sensitive. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

Troubleshooting AWS OpsWorks CM Identity and Access

Use the following information to help you diagnose and fix common issues that you might encounter when working with IAM. For troubleshooting information specific to OpsWorks CM, see Troubleshooting AWS OpsWorks for Chef Automate and Troubleshooting OpsWorks for Puppet Enterprise.

Topics

- I Am Not Authorized to Perform an Action in OpsWorks CM
- I Am Not Authorized to Perform iam:PassRole
- I Want to Allow People Outside of My AWS Account to Access My OpsWorks CM Resources

I Am Not Authorized to Perform an Action in OpsWorks CM

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your sign-in credentials.

The following example error occurs when user `mateojackson` tries to use the console to view details about an OpsWorks CM server, but does not have `opsworks-cm:DescribeServers` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 opsworks-cm:DescribeServers on resource: test-chef-automate-server
```

In this case, Mateo asks his administrator to update policies to allow him to access the `test-chef-automate-server` resource using the `opsworks-cm:DescribeServers` action.

I Am Not Authorized to Perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your sign-in credentials. Ask that person to update your policies to allow you to pass a role to OpsWorks CM.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when a user named `marymajor` tries to use the console to perform an action in OpsWorks CM. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I Want to Allow People Outside of My AWS Account to Access My OpsWorks CM Resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- OpsWorks CM supports granting users from more than one account access to manage an OpsWorks CM server.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

AWS managed policies for AWS OpsWorks Configuration Management

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

AWS managed policy: `AWSOpsWorksCMServiceRole` - Deprecated

You can attach `AWSOpsWorksCMServiceRole` to your IAM entities. OpsWorks CM also attaches this policy to a service role that allows OpsWorks CM to perform actions on your behalf.

This policy grants *`administrative`* permissions that allow OpsWorks CM administrators to create, manage, and delete OpsWorks CM servers and backups.

For more information, see Deprecated AWS managed policies in the *AWS Managed Policy Reference Guide*.

Permissions details

This policy includes the following permissions.

- `opsworks-cm` – Allows principals to delete existing servers, and start maintenance runs.
- `acm` – Allows principals to delete or import certificates from AWS Certificate Manager that let users connect to an OpsWorks CM server.
- `cloudformation` – Allows OpsWorks CM to create and manage AWS CloudFormation stacks when principals create, update, or delete OpsWorks CM servers.
- `ec2` – Allows OpsWorks CM to launch, provision, update, and terminate Amazon Elastic Compute Cloud instances when principals create, update, or delete OpsWorks CM servers.
- `iam` – Allows OpsWorks CM to create service roles that are required for creating and managing OpsWorks CM servers.
- `tag` – Allows principals to apply and remove tags from OpsWorks CM resources, including servers and backups.
- `s3` – Allows OpsWorks CM to create Amazon S3 buckets for storing server backups, manage objects in S3 buckets on principal request (for example, deleting a backup), and delete buckets.
- `secretsmanager` – Allows OpsWorks CM to create and manage Secrets Manager secrets, and apply or remove tags from secrets.

- `ssm` – Allows OpsWorks CM to use Systems Manager Run Command on the instances that are OpsWorks CM servers.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket*"
            ],
            "Action": [
                "s3:CreateBucket",
                "s3:DeleteObject",
                "s3:DeleteBucket",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:PutBucketPolicy",
                "s3:PutObject",
                "s3:GetBucketTagging",
                "s3:PutBucketTagging"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": [
                "*"
            ],
            "Action": [
                "tag:UntagResources",
                "tag:TagResources"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": [
                "*"
            ],
            "Action": [
                "ssm:DescribeInstanceInformation",
```

```
                "ssm:GetCommandInvocation",
                "ssm:ListCommandInvocations",
                "ssm:ListCommands"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringLike": {
                    "ssm:resourceTag/aws:cloudformation:stack-name": "aws-
opsworks-cm-*"
                }
            },
            "Action": [
                "ssm:SendCommand"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": [
                "arn:aws:ssm:*::document/*",
                "arn:aws:s3:::amzn-s3-demo-bucket*"
            ],
            "Action": [
                "ssm:SendCommand"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": [
                "*"
            ],
            "Action": [
                "ec2:AllocateAddress",
                "ec2:AssociateAddress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:CreateImage",
                "ec2:CreateSecurityGroup",
                "ec2:CreateSnapshot",
                "ec2:CreateTags",
                "ec2:DeleteSecurityGroup",
```

```
                "ec2:DeleteSnapshot",
                "ec2:DeregisterImage",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAddresses",
                "ec2:DescribeImages",
                "ec2:DescribeInstanceStatus",
                "ec2:DescribeInstances",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSnapshots",
                "ec2:DescribeSubnets",
                "ec2:DisassociateAddress",
                "ec2:ReleaseAddress",
                "ec2:RunInstances",
                "ec2:StopInstances"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringLike": {
                    "ec2:ResourceTag/aws:cloudformation:stack-name": "aws-
opsworks-cm-*"
                }
            },
            "Action": [
                "ec2:TerminateInstances",
                "ec2:RebootInstances"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": [
                "arn:aws:opsworks-cm:*:*:server/*"
            ],
            "Action": [
                "opsworks-cm:DeleteServer",
                "opsworks-cm:StartMaintenance"
            ]
        },
        {
            "Effect": "Allow",
```

```
            "Resource": [
                "arn:aws:cloudformation:*:*:stack/aws-opsworks-cm-*"
            ],
            "Action": [
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStackEvents",
                "cloudformation:DescribeStackResources",
                "cloudformation:DescribeStacks",
                "cloudformation:UpdateStack"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iam::*:role/aws-opsworks-cm-*",
                "arn:aws:iam::*:role/service-role/aws-opsworks-cm-*"
            ],
            "Action": [
                "iam:PassRole"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": [
                "acm:DeleteCertificate",
                "acm:ImportCertificate"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": "arn:aws:secretsmanager:*:*:opsworks-cm!aws-opsworks-cm-
secrets-*",
            "Action": [
                "secretsmanager:CreateSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:UpdateSecret",
                "secretsmanager:DeleteSecret",
                "secretsmanager:TagResource",
                "secretsmanager:UntagResource"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": "ec2:DeleteTags",
            "Resource": [
                "arn:aws:ec2:*:*:instance/*",
                "arn:aws:ec2:*:*:elastic-ip/*",
                "arn:aws:ec2:*:*:security-group/*"
            ]
        }
    ]
}
```

AWS managed policy: AWSOpsWorksCMInstanceProfileRole - Deprecated

You can attach `AWSOpsWorksCMInstanceProfileRole` to your IAM entities. OpsWorks CM also attaches this policy to a service role that allows OpsWorks CM to perform actions on your behalf.

This policy grants *`administrative`* permissions that allow the Amazon EC2 instances that are used as OpsWorks CM servers to get information from AWS CloudFormation and AWS Secrets Manager, and store server backups in Amazon S3 buckets.

For more information, see Deprecated AWS managed policies in the *AWS Managed Policy Reference Guide*.

Permissions details

This policy includes the following permissions.

- `acm` – Allows OpsWorks CM server EC2 instances to get certificates from AWS Certificate Manager that let users connect to an OpsWorks CM server.
- `cloudformation` – Allows OpsWorks CM server EC2 instances to get information about AWS CloudFormation stacks during the instance creation or update process, and send signals to AWS CloudFormation about its status.
- `s3` – Allows OpsWorks CM server EC2 instances to upload and store server backups in S3 buckets, stop or roll back uploads if necessary, and delete backups from S3 buckets.
- `secretsmanager` – Allows OpsWorks CM server EC2 instances to get the values of OpsWorks CM related Secrets Manager secrets.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "cloudformation:DescribeStackResource",
                "cloudformation:SignalResource"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        },
        {
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:DeleteObject",
                "s3:GetObject",
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "s3:ListMultipartUploadParts",
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket*",
            "Effect": "Allow"
        },
        {
            "Action": "acm:GetCertificate",
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:*:*:opsworks-cm!aws-opsworks-cm-
secrets-*",
            "Effect": "Allow"
        }
    ]
}
```

OpsWorks CM updates to AWS managed policies

View details about updates to AWS managed policies for OpsWorks CM since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the OpsWorks CM Document history page.

Change	Description	Date
AWSOpsWorksCMInstanceProfileRole - Deprecated	This policy has been deprecated because the service is deprecated.	May 26, 2025
AWSOpsWorksCMServiceRole - Deprecated	This policy has been deprecated because the service is deprecated.	May 26, 2025
AWSOpsWorksCMServiceRole - Updated managed policy	OpsWorks CM updated the managed policy that allows OpsWorks CM administrators to create, manage, and delete OpsWorks CM servers and backups. The change adds `opsworks-cm!` to the resource name for Secrets Manager secrets, so that OpsWorks CM is allowed to own the secrets.	April 23, 2021
OpsWorks CM started tracking changes	OpsWorks CM started tracking changes for its AWS managed policies.	April 23, 2021

Cross-service confused deputy prevention in AWS OpsWorks CM

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur

when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that AWS OpsWorks CM gives another service to the resource. If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both global condition context keys to limit permissions. If you use both global condition context keys and the `aws:SourceArn` value contains the account ID, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement. Use `aws:SourceArn` if you want only one resource to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

The value of `aws:SourceArn` must be the ARN of an OpsWorks CM Chef or Puppet server.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the AWS OpsWorks CM server. If you don't know the full ARN, or if you are specifying multiple server ARNs, use the `aws:SourceArn` global context condition key with wildcards (*) for the unknown portions of the ARN. For example, `arn:aws:`*`servicename`*`:*:`*`123456789012`*`:*`.

The following section shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in AWS OpsWorks CM to prevent the confused deputy problem.

Prevent confused deputy exploits in AWS OpsWorks CM

This section describes how you can help prevent confused deputy exploits in AWS OpsWorks CM, and includes examples of permissions policies that you can attach to the IAM role you are using to access AWS OpsWorks CM. As a security best practice, we recommend adding the `aws:SourceArn` and `aws:SourceAccount` condition keys to the trust relationships your IAM role has with other services. The trust relationships allow AWS OpsWorks CM to assume a role to perform actions in other services that are required to create or manage your AWS OpsWorks CM servers.

To edit trust relationships to add `aws:SourceArn` and `aws:SourceAccount` condition keys

1. Open the IAM console at https://console.aws.amazon.com/iam/.

2. In the left navigation pane, choose **Roles**.
3. In the **Search** box, search for the role that you use for access to AWS OpsWorks CM. The AWS managed role is `aws-opsworks-cm-service-role`.
4. On the **Summary** page for the role, choose the **Trust relationships** tab.
5. On the **Trust relationships** tab, choose **Edit trust relationship**.
6. In the **Policy document**, add at least one of the `aws:SourceArn` or `aws:SourceAccount` condition keys to the policy. Use `aws:SourceArn` to restrict the trust relationship between cross services (such as AWS Certificate Manager and Amazon EC2) and AWS OpsWorks CM to specific AWS OpsWorks CM servers, which is more restrictive. Add `aws:SourceAccount` to restrict the trust relationship between cross services and AWS OpsWorks CM to servers in a specific account, which is less restrictive. The following is an example. Note that if you use both condition keys, the account IDs must be the same.

   JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "opsworks-cm.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:opsworks-cm:us-
east-2:123456789012:server/my-opsworks-server/EXAMPLEabcd-1234-efghEXAMPLE-
ID"
        }
      }
    }
  ]
}
```

7. When you are finished adding condition keys, choose **Update trust policy**.

The following are additional examples of roles that limit access to AWS OpsWorks CM servers by using `aws:SourceArn` and `aws:SourceAccount`.

Topics

- Example: Accessing AWS OpsWorks CM servers in a specific region
- Example: Adding more than one server ARN to aws:SourceArn

Example: Accessing AWS OpsWorks CM servers in a specific region

The following role trust relationship statement accesses any AWS OpsWorks CM servers in the US East (Ohio) Region (`us-east-2`). Note that the region is specified in the ARN value of `aws:SourceArn`, but the server ID value is a wildcard (*).

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "opsworks-cm.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:opsworks-cm:us-east-2:123456789012:server/*"
        }
      }
    }
  ]
}
```

Example: Adding more than one server ARN to `aws:SourceArn`

The following example limits access to an array of two AWS OpsWorks CM servers in account ID 123456789012.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "opsworks-cm.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnEquals": {
          "aws:SourceArn": [
             "arn:aws:opsworks-cm:us-east-2:123456789012:server/my-chef-
server/unique_ID",
             "arn:aws:opsworks-cm:us-east-2:123456789012:server/my-puppet-
server/unique_ID"
           ]
       }
      }
    }
  ]
}
```

Internetwork Traffic Privacy

OpsWorks CM uses the same transmission security protocols generally used by AWS: HTTPS, or HTTP with TLS encryption.

Logging and Monitoring in OpsWorks CM

OpsWorks CM logs all API actions to CloudTrail. For more information, see the following topics:

- Logging OpsWorks for Puppet Enterprise API Calls with AWS CloudTrail
- Logging AWS OpsWorks for Chef Automate API Calls with AWS CloudTrail

Compliance Validation for OpsWorks CM

OpsWorks CM supports the following compliance programs and regulations:

- Payment Card Industry (PCI)
- Health Insurance Portability and Accountability Act of 1996 (HIPAA)
- AWS System and Organization Controls (SOC) 1, 2, and 3
- General Data Protection Regulation (GDPR)

Third-party auditors assess the security and compliance of OpsWorks CM as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using OpsWorks CM is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.

- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in OpsWorks CM

OpsWorks CM enables daily backups of servers by default when you create a server. Backups are encrypted and are stored in an Amazon S3 bucket. By default, this bucket is accessible only to the account that created the server. You can add bucket access for other users or configure cross-region backups in Amazon S3 at your discretion. Chef and Puppet support cross-region encryption, because both products encrypt traffic between your OpsWorks CM server and managed nodes.

OpsWorks CM does not support high availability (HA) configurations.

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about how to back up and restore servers in OpsWorks CM, see the following:

- Back Up and Restore an OpsWorks for Puppet Enterprise Server
- Back Up and Restore an AWS OpsWorks for Chef Automate Server

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure Security in AWS OpsWorks CM

As a managed service, AWS OpsWorks Configuration Management is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access OpsWorks CM through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

OpsWorks CM does not support private link or VPC private endpoints.

OpsWorks CM does not support resource-based policies. For more information, see AWS Services that Work with IAM in the *AWS Identity and Access Management User Guide*.

Configuration and Vulnerability Analysis in OpsWorks CM

OpsWorks CM performs periodic kernel and security updates to the operating system that is running on your OpsWorks CM server. Users can set a window of time for automatic updates to occur for up to two weeks from the current date. OpsWorks CM pushes automatic updates of Chef and Puppet Enterprise minor versions. For more information about configuring updates for AWS OpsWorks for Chef Automate, see System Maintenance (Chef) in this guide. For more information about configuring updates for OpsWorks for Puppet Enterprise, see System Maintenance (Puppet) in this guide.

Security Best Practices for OpsWorks CM

OpsWorks CM, like all AWS services, offers security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

- **Secure your Starter Kit and downloaded login credentials.** When you create a new OpsWorks CM server or download a new Starter Kit and credentials from the OpsWorks CM console, store these items in a secure location that requires at least one factor of authentication at minimum. The credentials provide administrator-level access to your server.

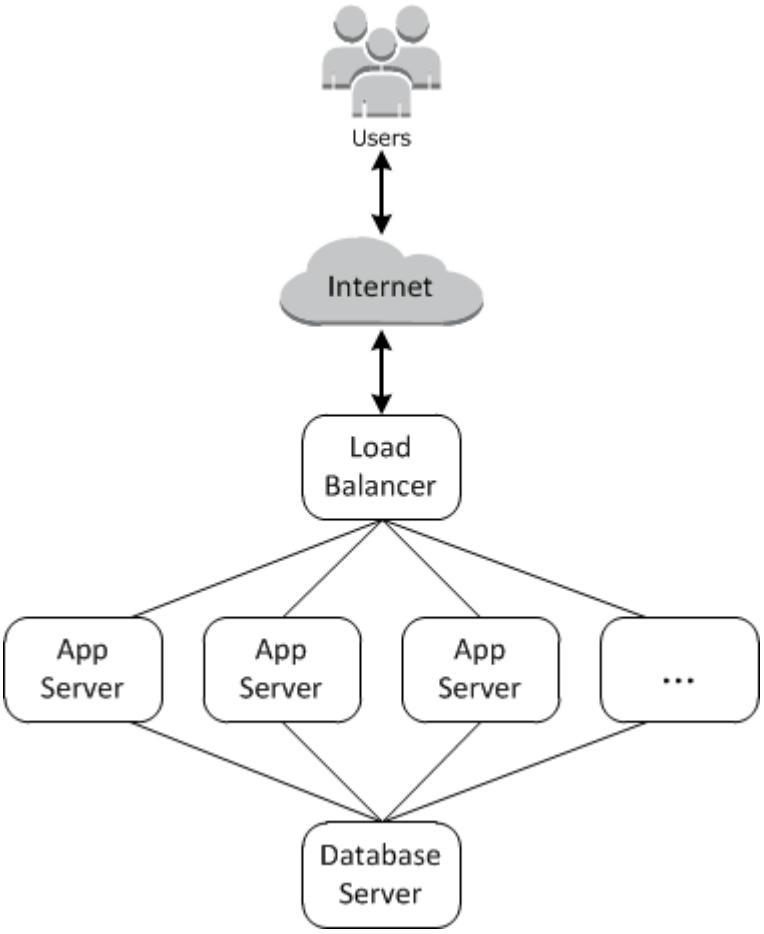
- **Secure your configuration code.** Secure your Chef or Puppet configuration code (cookbooks and modules) using recommended protocols for your source repositories. For example, you can restrict permissions to repositories in AWS CodeCommit, or follow guidelines on the GitHub website for securing GitHub repositories.
- **Use CA-signed certificates to connect to nodes.** Although you can use self-signed certificates when you are registering or bootstrapping nodes on your OpsWorks CM server, as a best practice, use CA-signed certificates. We recommend that you use a certificate signed by a certificate authority (CA).
- **Do not share Chef or Puppet management console sign-in credentials** with other users. An administrator should create separate users for each user of the Chef or Puppet console websites.
  - Manage Users in Chef Automate
  - Manage Users in Puppet Enterprise
- **Configure automatic backups and system maintenance updates.** Configuring automatic maintenance updates on your OpsWorks CM server helps ensure that your server is running the most current security-related operating system updates. Configuring automatic backups helps ease disaster recovery and speed restoration time in the event of an incident or failure. Limit access to the Amazon S3 bucket that stores your OpsWorks CM server backups; do not grant access to **Everyone**. Grant read or write access to other users individually as needed, or create a security group in IAM for those users, and assign access to the security group.
  - System Maintenance (Chef)
  - System Maintenance (Puppet)
  - Back Up and Restore an AWS OpsWorks for Chef Automate Server
  - Back Up and Restore an OpsWorks for Puppet Enterprise Server
  - Creating Your First IAM Delegated User and Group in the *AWS Identity and Access Management User Guide*
  - Security Best Practices for Amazon S3 in the *Amazon Simple Storage Service Developer Guide*

OpsWorks Stacks

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

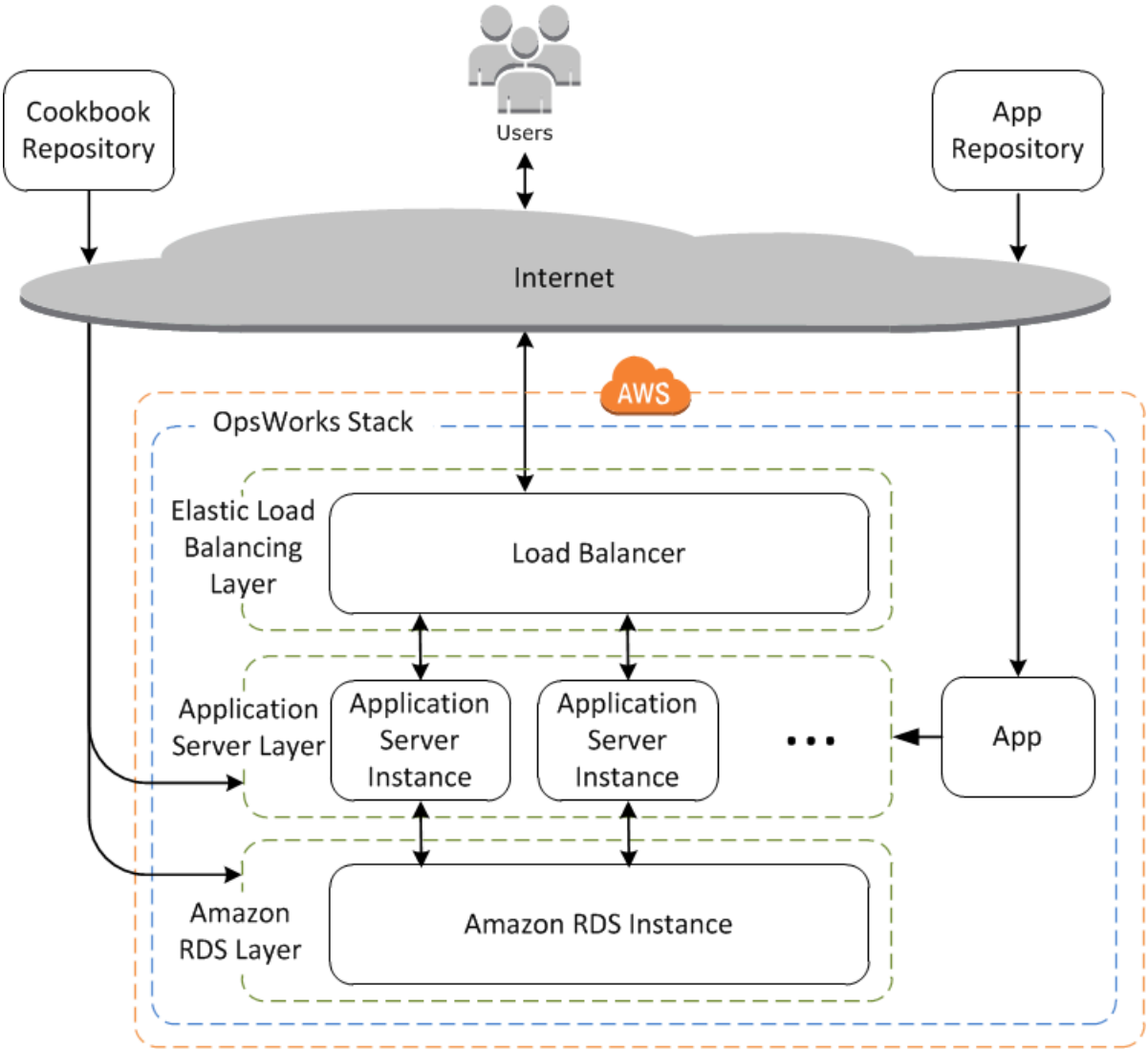
Cloud-based computing usually involves groups of AWS resources, such as Amazon EC2 instances and Amazon Relational Database Service (RDS) instances, which must be created and managed collectively. For example, a web application typically requires application servers, database servers, load balancers, and so on. This group of instances is typically called a *stack*; a simple application server stack might look something like the following.



In addition to creating the instances and installing the necessary packages, you typically need a way to distribute applications to the application servers, monitor the stack's performance, manage security and permissions, and so on.

OpsWorks Stacks provides a simple and flexible way to create and manage stacks and applications.

Here's how a basic application server stack might look with OpsWorks Stacks. It consists of a group of application servers running behind an Elastic Load Balancing load balancer, with a backend Amazon RDS database server.



Although relatively simple, this stack shows all the key OpsWorks Stacks features. Here's how it's put together.

Topics

- Stacks

- Layers
- Recipes and LifeCycle Events
- Instances
- Apps
- Customizing your Stack
- Resource Management
- Security and Permissions
- Monitoring and Logging
- CLI, SDK, and AWS CloudFormation Templates
- AWS OpsWorks Stacks End of Life FAQs
- Migrating your AWS OpsWorks Stacks applications to AWS Systems Manager Application Manager
- Using the AWS OpsWorks Stacks Detach in Place tool
- Getting Started with OpsWorks Stacks
- OpsWorks Stacks Best Practices
- Stacks
- Layers
- Instances
- Apps
- Cookbooks and Recipes
- Resource Management
- Tags
- Monitoring
- Security and Permissions
- OpsWorks Stacks Support for Chef 12 Linux
- Support for Previous Chef Versions in OpsWorks Stacks
- Using OpsWorks Stacks with Other AWS Services
- Using the OpsWorks Stacks CLI
- Debugging and Troubleshooting Guide
- OpsWorks Stacks Agent CLI

- OpsWorks Stacks Data Bag Reference
- OpsWorks Agent Changes

Stacks

The *stack* is the core OpsWorks Stacks component. It is basically a container for AWS resources—Amazon EC2 instances, Amazon RDS database instances, and so on—that have a common purpose and should be logically managed together. The stack helps you manage these resources as a group and also defines some default configuration settings, such as the instances' operating system and AWS region. If you want to isolate some stack components from direct user interaction, you can run the stack in a VPC.

Layers

You define the stack's constituents by adding one or more *layers*. A layer represents a set of Amazon EC2 instances that serve a particular purpose, such as serving applications or hosting a database server.

You can customize or extend layers by modifying packages' default configurations, adding Chef recipes to perform tasks such as installing additional packages, and more.

For all stacks, OpsWorks Stacks includes *service layers*, which represent the following AWS services.

- Amazon Relational Database Service
- Elastic Load Balancing
- Amazon Elastic Container Service

Layers give you complete control over which packages are installed, how they are configured, how applications are deployed, and more.

Recipes and LifeCycle Events

Layers depend on Chef recipes to handle tasks such as installing packages on instances, deploying apps, running scripts, and so on. One of the key OpsWorks Stacks features is a set of *lifecycle events*—Setup, Configure, Deploy, Undeploy, and Shutdown—which automatically run a specified set of recipes at the appropriate time on each instance.

Each layer can have a set of recipes assigned to each lifecycle event, which handle a variety of tasks for that event and layer. For example, after an instance that belongs to a web server layer finishes booting, OpsWorks Stacks does the following.

1. Runs the layer's Setup recipes, which could perform tasks such as installing and configuring a web server.
2. Runs the layer's Deploy recipes, which deploy the layer's applications from a repository to the instance and perform related tasks, such as restarting the service.
3. Runs the Configure recipes on every instance in the stack so each instance can adjust its configuration as needed to accommodate the new instance.

   For example, on an instance running a load balancer, a Configure recipe could modify the load balancer's configuration to include the new instance.

If an instance belongs to multiple layers, OpsWorks Stacks runs the recipes for each layer so you can, for example, have an instance that supports a PHP application server and a MySQL database server.

If you have implemented recipes, you can assign each recipe to the appropriate layer and event and OpsWorks Stacks automatically runs them for you at the appropriate time. You can also run recipes manually, at any time.

Instances

An *instance* represents a single computing resource, such as an Amazon EC2 instance. It defines the resource's basic configuration, such as operating system and size. Other configuration settings, such as Elastic IP addresses or Amazon EBS volumes, are defined by the instance's layers. The layer's recipes complete the configuration by performing tasks such as installing and configuring packages and deploying apps.

You can use OpsWorks Stacks to create instances and add them to a layer. When you start the instance, OpsWorks Stacks launches an Amazon EC2 instance using the configuration settings specified by the instance and its layer. After the Amazon EC2 instance has finished booting, OpsWorks Stacks installs an agent that handles communication between the instance and the service and runs the appropriate recipes in response to lifecycle events.

OpsWorks Stacks supports the following instance types, which are characterized by how they are started and stopped.

- **24/7 instances** are started manually and run until you stop them.
- **Time-based instances** are run by OpsWorks Stacks on a specified daily and weekly schedule.

  They allow your stack to automatically adjust the number of instances to accommodate predictable usage patterns.
- **Load-based instances** are automatically started and stopped by OpsWorks Stacks, based on specified load metrics, such as CPU utilization.

  They allow your stack to automatically adjust the number of instances to accommodate variations in incoming traffic. Load-based instances are available only for Linux-based stacks.

OpsWorks Stacks supports instance autohealing. If an agent stops communicating with the service, OpsWorks Stacks automatically stops and restarts the instance.

You can also incorporate Linux-based computing resources into a stack that was created outside of OpsWorks Stacks.

- Amazon EC2 instances that you created directly by using the Amazon EC2 console, CLI, or API.
- *On-premises* instances running on your own hardware, including instances running in virtual machines.

After you have registered one of these instances, it becomes an OpsWorks Stacks instance and you can manage it in much the same way as instances that you create with OpsWorks Stacks.

Apps

You store applications and related files in a repository, such as an Amazon S3 bucket. Each application is represented by an *app*, which specifies the application type and contains the information that is needed to deploy the application from the repository to your instances, such as the repository URL and password. When you deploy an app, OpsWorks Stacks triggers a Deploy event, which runs the Deploy recipes on the stack's instances.

You can deploy apps in the following ways:

- Automatically—When you start instances, OpsWorks Stacks automatically runs the instance's Deploy recipes.

- Manually—If you have a new app or want to update an existing one, you can manually run the online instances' Deploy recipes.

You typically have OpsWorks Stacks run the Deploy recipes on the entire stack, which allows the other layers' instances to modify their configuration appropriately. However, you can limit deployment to a subset of instances if, for example, you want to test a new app before deploying it to every app server instance.

Customizing your Stack

OpsWorks Stacks provides a variety of ways to customize layers to meet your specific requirements:

- You can modify how OpsWorks Stacks configures packages by overriding attributes that represent the various configuration settings, or by even overriding the templates used to create configuration files.
- You can extend an existing layer by providing your own recipes to perform tasks such as running scripts or installing and configuring nonstandard packages.

All stacks can include one or more layers, which start with only a minimal set of recipes. You add functionality to the layer by implementing recipes to handle tasks such as installing packages, deploying apps, and so on. You package your custom recipes and related files in one or more *cookbooks* and store the cookbooks in a repository such Amazon S3 or Git.

You can run recipes manually, but OpsWorks Stacks also lets you automate the process by supporting a set of five *lifecycle events*:

- **Setup** occurs on a new instance after it successfully boots.
- **Configure** occurs on all of the stack's instances when an instance enters or leaves the online state.
- **Deploy** occurs when you deploy an app.
- **Undeploy** occurs when you delete an app.
- **Shutdown** occurs when you stop an instance.

Each layer can have any number of recipes assigned to each event. When a lifecycle event occurs on a layer's instance, OpsWorks Stacks runs the associated recipes. For example, when a Deploy event

occurs on an app server instance, OpsWorks Stacks runs the layer's Deploy recipes to download the app or perform related tasks.

Resource Management

You can incorporate other AWS resources, such as Elastic IP addresses, into your stack. You can use the OpsWorks Stacks console or API to register resources with a stack, attach registered resources to or detach them from instances, and move resources from one instance to another.

Security and Permissions

AWS OpsWorks Stacks integrates with AWS Identity and Access Management (IAM) to provide robust ways of controlling how users access OpsWorks Stacks, including the following:

- How individual users can interact with each stack, such as whether they can create stack resources such as layers and instances, or whether they can use SSH or RDP to connect to a stack's Amazon EC2 instances.
- How OpsWorks Stacks can act on your behalf to interact with AWS resources such as Amazon EC2 instances.
- How apps that run on OpsWorks Stacks instances can access AWS resources such as Amazon S3 buckets.
- How to manage users' public SSH keys and RDP passwords and connect to an instance.

Monitoring and Logging

OpsWorks Stacks provides several features to help you monitor your stack and troubleshoot issues with your stack and any recipes. For all stacks:

- OpsWorks Stacks provides a set of custom CloudWatch metrics for Linux stacks, which are summarized for your convenience on the **Monitoring** page.

  OpsWorks Stacks supports the standard CloudWatch metrics for Windows stacks. You can monitor them with the CloudWatch console.
- CloudTrail logs, which record API calls made by or on behalf of OpsWorks Stacks in your AWS account.
- An event log, which lists all events in your stack.

- Chef logs that detail what transpired for each lifecycle event on each instance, such as which recipes were run and which errors occurred.

Linux-based stacks can also include a Ganglia master layer, which you can use to collect and display detailed monitoring data for the instances in your stack.

CLI, SDK, and AWS CloudFormation Templates

In addition to the console, OpsWorks Stacks also supports a command-line interface (CLI) and SDKs for multiple languages that can be used to perform any operation. Consider these features:

- The OpsWorks Stacks CLI is part of the AWS CLI, and can be used to perform any operation from the command-line.

  The AWS CLI supports multiple AWS services and can be installed on Windows, Linux, or OS X systems.
- OpsWorks Stacks is included in AWS Tools for Windows PowerShell and can be used to perform any operation from a Windows PowerShell command line.
- The OpsWorks Stacks SDK is included in the AWS SDKs, which can be used by applications implemented in: Java, JavaScript (browser-based and Node.js), .NET, PHP, Python (boto), or Ruby.

You can also use AWS CloudFormation templates to provision stacks. For some examples, see AWS OpsWorks Snippets.

AWS OpsWorks Stacks End of Life FAQs

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible.

Topics

- How will existing customers be affected by this End of Life?

- Is AWS OpsWorks Stacks accepting new customers?
- Where should I migrate my existing stacks to?
- How can I keep my existing Amazon EC2 instances after End of Life?
- Will the End of Life affect all AWS Regions at the same time?
- What level of technical support is available for AWS OpsWorks Stacks?
- Will there be any new feature releases for AWS OpsWorks Stacks?

How will existing customers be affected by this End of Life?

Existing customers will be unaffected until May 26, 2024, the End of Life date for AWS OpsWorks Stacks. After May 26, 2024, customers will be unable to use the OpsWorks console, API, CLI, and CloudFormation resources.

Is AWS OpsWorks Stacks accepting new customers?

No. AWS OpsWorks Stacks is no longer accepting new customers and only existing customers are able to create new stacks at this time.

Where should I migrate my existing stacks to?

We recommend AWS OpsWorks Stacks customers migrate their workloads to AWS Systems Manager where they can take advantage of the following features:

- Modern Chef versions
- SSM Agent
- Application Load Balancers
- Enhanced scaling features via Auto Scaling groups
- Ability to define desired host characteristics using EC2 launch templates
- Newer instance types
- Newer EBS volume types

For information about Systems Manager, see the AWS Systems Manager User Guide. For information about migrating to AWS Systems Manager, see Migrating your AWS OpsWorks Stacks applications to AWS Systems Manager Application Manager

How can I keep my existing Amazon EC2 instances after End of Life?

After the End of Life date is reached your Amazon EC2 instances will remain in your account, but you'll no longer be able to use the OpsWorks Stacks service to control and manage the instances.

You can use the AWS OpsWorks Stacks Detach in Place tool to detach your OpsWorks instances from the OpsWorks Stacks service. After detachment, you can use Amazon EC2, AWS Systems Manager, or any EC2 compatible approach to configure and manage the instances. For more information, see Using the AWS OpsWorks Stacks Detach in Place tool.

Will the End of Life affect all AWS Regions at the same time?

Yes. The OpsWorks console, API, CLI, and CloudFormation resources will be discontinued in all AWS Regions simultaneously on May 26, 2024. For a list of AWS Regions where AWS OpsWorks Stacks is available, see AWS Regional Services List.

What level of technical support is available for AWS OpsWorks Stacks?

AWS will continue to provide the same level of support for AWS OpsWorks Stacks that customers have today up until the End of Life date. If you have questions or concerns, you can contact the AWS Support Team on AWS re:Post or through AWS Premium Support.

Will there be any new feature releases for AWS OpsWorks Stacks?

No. As the service is reaching End of Life, we will not release any new features. However, we will continue to make security improvements and manage Amazon EC2 instances as expected until the End of Life date.

Migrating your AWS OpsWorks Stacks applications to AWS Systems Manager Application Manager

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible.

You can now migrate your AWS OpsWorks Stacks applications to Application Manager, a capability of AWS Systems Manager, using a migration script. Migrating your Stacks applications to Systems Manager Application Manager allows you to use AWS features that are not available in AWS OpsWorks Stacks, such as new Amazon EC2 instance types like Graviton, new Amazon Elastic Block Store (EBS) volumes like gp3, new operating systems, integrations with Auto Scaling groups, and application load balancers.

With this release, you can now monitor and run operations on your migrated instances using a new **Instances** tab available from Systems Manager Application Manager. You can use the **Instances** tab to view multiple AWS instances in one place. Using this tab, you can view information about instance health and troubleshoot issues. For more information about working with the **Instances** tab, see Working with your application instances in the *AWS Systems Manager User Guide*.

Topics

- How the script works
- Prerequisites
- Limitations
- Getting started
- FAQ
- Troubleshooting

How the script works

OpsWorks provides a script that you can run to migrate your AWS OpsWorks Stacks applications to Systems Manager Application Manager using a CloudFormation template. The script gets information about an existing OpsWorks layer and depending on the value of the `--provision-application` parameter for the script, either provisions a clone of your application, or provides a starter CloudFormation template that you can modify using AWS CloudFormation.

Prerequisites

- Be sure that the AWS CLI is installed and configured. For more information about installing the AWS CLI, see Installing or updating the latest version of the AWS CLI in the *AWS Command Line Interface User Guide*.

> **Note**
>
> If you do not want to configure the AWS CLI, you can also run commands using AWS CloudShell. For more information about working with CloudShell, see Working with AWS CloudShell in the *AWS CloudShell User Guide*.

- Be sure that Python version 3.6 or newer is installed or comes with the Amazon Machine Image (AMI).
- Be sure that your operating system is supported. You can download and run the migration script on the following operating systems.
  - Amazon Linux and Amazon Linux 2
  - Ubuntu 18.04 LTS, 20.04 LTS, 22.04 LTS
  - Red Hat Enterprise Linux 8
  - Windows Server 2019, Windows 10 Enterprise

> **Note**
>
> Windows Server 2022 is not supported.

Limitations

The new OpsWorks architecture differs from the architecture for AWS OpsWorks Stacks. This section describes the known limitations of this architecture.

The following are not supported by the new OpsWorks architecture.

- Running Chef recipes on Windows and CentOS instances
- Built-in Chef 11 layers and Berkshelf
- Chef attributes and data bags
- On-premises instances
- Instances imported from EC2
- No support for installing a user specified list of operating system packages
- Apps are not supported or migrated

The following are supported with limitations.

- The migration script clones EBS volume information, but excludes mount points and actual data contained in the volumes.
- Time-based and load-based scaled instances are migrated, but any scaling rules associated with these instances are not migrated. You can modify the Auto Scaling group to achieve similar results.
- IAM entities defined in the stack's **Permissions** page in the OpsWorks console are not created or generated.
- The migration script is only capable of provisioning single layer applications in Systems Manager. For example, if you run the script twice for two layers in the same stack, you get two different applications in Systems Manager.

Getting started

The migration script, `stack_exporter.py`, is a Python script that you can run locally, or on an EC2 instance. Before running the script, be sure all prerequisites are met. For more about the prerequisites, see Prerequisites.

The steps in the following sections show you how to migrate your OpsWorks stacks to Systems Manager Application Manager.

Topics

- Step 1: Prepare your environment for running the script
- Step 2: Download the migration script
- Step 3: Set up your environment to run the script
- Step 4: Run the script
- Step 5: Provision a CloudFormation stack
- Step 6: Review the provisioned resources
- Step 7: Start an instance
- Step 8: Review the instance
- Step 9: Monitor and run operations on your instances using Systems Manager Application Manager

Step 1: Prepare your environment for running the script

Prepare your environment by running the appropriate commands for your operating system.

Topics

- Amazon Linux 2
- Amazon Linux
- Ubuntu 18.04, 20.04, 22.04
- Red Hat Enterprise Linux 8
- Windows Server 2019, Windows 10 Enterprise

Amazon Linux 2

```
sudo su
python3 -m pip install pipenv
PATH="$PATH:/usr/local/bin"
yum update
yum install git
```

Amazon Linux

```
sudo su
PATH="$PATH:/usr/local/bin"
export LC_ALL=en_US.utf-8
export LANG=en_US.utf-8
yum update
yum list | grep python3
yum install python36 // Any python version
yum install git
```

For Python version 3.6, also run:

```
python3 -m pip install pipenv==2022.4.8
```

For Python version 3.7 and newer, also run:

```
python3 -m pip install pipenv
```

Ubuntu 18.04, 20.04, 22.04

```
sudo su
export PATH="${HOME}/.local/bin:$PATH"
apt-get update
apt install python3-pip
apt-get install git // if git is not installed
python3 -m pip install --user pipenv==2022.4.8
```

Red Hat Enterprise Linux 8

```
sudo su
sudo dnf install python3
PATH="$PATH:/usr/local/bin"
yum update
yum install git
python3 -m pip install pipenv==2022.4.8
```

Windows Server 2019, Windows 10 Enterprise

> **Note**
>
> For Windows Server 2019, install Python version 3.6.1 or newer.

```
pip install pipenv
```

If Git is not already installed, download and install Git.

If you use Git as a cookbook source, add your Git server to a `known_hosts` file before running the script on Windows. You can use PowerShell to create the following function.

```
function add_to_known_hosts($server){
    $new_host=$(ssh-keyscan $server 2> $null)
    $existing_hosts=''
    if (!(test-path "$env:userprofile\.ssh")) {
        md "$env:userprofile\.ssh"
    }
    if ((test-path "$env:userprofile\.ssh\known_hosts")) {
        $existing_hosts=Get-Content "$env:userprofile\.ssh\known_hosts"
```

```
    }
    $host_added=0
    foreach ($line in $new_host) {
        if (!($existing_hosts -contains $line)) {
            Add-Content -Path "$env:userprofile\.ssh\known_hosts" -Value $line
            $host_added=1
    }
   }
   if ($host_added) {
       echo "$server has been added to known_hosts."
   } else {
       echo "$server already exists in known_hosts."
   }
}
```

You can then provide your Git server (for example, github.com, git-codecommit.*repository_region*.amazonaws.com) when you run the function.

```
add_to_known_hosts "myGitServer"
```

Step 2: Download the migration script

Download the zip file containing the migration script and all relevant files by running the following command.

```
aws s3api get-object \
    --bucket export-opsworks-stacks-bucket-prod-us-east-1 \
    --key export_opsworks_stacks_script.zip export_opsworks_stacks_script.zip
```

If you are using Linux, install the unzip utility using the following commands.

```
sudo apt-get install unzip
sudo yum install unzip
```

Unzip the files using the appropriate command for your operating system.

For Linux, use the following command.

```
unzip export_opsworks_stacks_script.zip
```

For Windows, use the `Expand-Archive` command in PowerShell.

```
Expand-Archive -LiteralPath PathToZipFile -DestinationPath PathToDestination
```

After the file is unzipped, the following directories and files are available.

- README.md
- LICENSE
- NOTICE
- requirements.txt
- templates/
  - OpsWorksCFNTemplate.yaml
  - MountEBSVolumes.yaml
- opsworks/
- cloudformation/
- instances_tab/
- cfn_stack_deployer.py
- s3.py
- stack_exporter_context.py
- stack_exporter.py

Step 3: Set up your environment to run the script

Set up your environment to the run the script by using the following command.

```
pipenv install -r requirements.txt
pipenv shell
```

> **Note**
>
> Currently, the script can only provision single-layer applications in Application Manager. For example, if you run the script twice for two layers in the same stack, the script creates two different applications in Application Manager.

After setting up your environment, review the script parameters. You can view the available options for the migration script by running the `python3 stack_exporter.py --help` command.

Parameter	Description	Required	Type	Default value
`--layer-id`	Exports a CloudFormation template for this OpsWorks layer ID.	Yes	string	
`--region`	The AWS Region for the OpsWorks stack. If your OpsWorks stack Region and API endpoint Region are different , use the stack Region. This is the same Region as the other resources part of your OpsWorks stack (for example, EC2 instances and subnets).	No	string	us-east-1
`--provision-application`	By default, the script provisions the application exported by the CloudFormation template. Pass this parameter into the script with a value of FALSE to skip provisioning of the CloudFormation template.	No	Boolean	`TRUE`
`--launch-template`	This parameter defines whether to use an existing launch template, or create a new launch template. You can create a new launch template that uses the recommended instance properties, or that uses instance properties that match an online instance. Valid values include:	No	string	`RECOMMENDED`

Parameter	Description	Required	Type	Default value
	• `RECOMMENDED` - Uses instance characteristics from the latest AMI for the OpsWorks stack's OS and a c5.large instance size. • `MATCH_LAST_INSTANCE` - Uses latest available online instance characteristics. • *`LaunchTemplateID`* `/` [*`LaunchTemplateVersion`*] - Uses an existing launch template. Optionally, you can provide a template version. If you do not provide a template version, the script uses the default version.			

Parameter	Description	Required	Type	Default value
`--system-updates`	Defines whether to perform kernel and package updates when the instance boots. Valid values include: • `ALL_UPDATES` - Performs system updates for the kernel and packages when the instance boots. • `NO_UPDATES` - Does not perform system updates when the instance boots. • `MATCH_LAYER_SETTINGS` - Uses the OpsWorks layer's or instance's `InstallUpdatesOnBoot` property to determine whether to install system updates.	No	string	`ALL_UPDATES`
`--http-username`	The name of the Systems Manager `SecureString` parameter which stores the user name used to authenticate to the HTTP archive that contains the custom cookbooks.	No	string	
`--http-password`	The name of the Systems Manager `SecureString` parameter which stores the password used to authenticate to the HTTP archive that contains the custom cookbooks.	No	string	

Parameter	Description	Required	Type	Default value
`--repo-private-key`	The name of the Systems Manager `SecureString` parameter which stores the SSH key used to authenticate to the repository that contains the custom cookbooks. If the repository is on GitHub, you must generate a new `Ed25519` SSH key. If you do not generate a new `Ed25519` SSH key, the connection to the GitHub repository fails.	No	string	
`--lb-type`	The type of load balancer, if any, to create when migrating your existing load balancer. Valid values include: • `ALB` (Application Load Balancer) • `Classis` (Classic Load Balancer) • None (if you do not want to create a load balancer)	No	string	`ALB`

Parameter	Description	Required	Type	Default value
`--lb-access-logs-path`	The path to an existing S3 bucket and prefix for storing the load balancer access logs. The S3 bucket and load balancer must be in the same Region. If you do not provide a value and the `--lb-type` parameter value is set to None, the script creates a new S3 bucket and prefix. Be sure there is an appropriate bucket policy for this prefix.	No	string	
`--enable-instance-protection`	If set to TRUE , the script creates a custom termination policy (Lambda function) for your Auto Scaling group. EC2 instances with a `protected_instance` tag are protected from scale-in events. Add a `protected_instance` tag to each EC2 instance that you want to protect from scale-in events.	No	Boolean	FALSE
`--command-logs-bucket`	The name of an existing S3 bucket to store the `AWSApplyChefRecipe` and `MountEBSVolumes` logs. If you do not provide a value, the script creates a new S3 bucket.	No	string	`aws-opsworks-application-manager-logs-` *account-id*

Parameter	Description	Required	Type	Default value
`--custom-json-bucket`	The name of an existing S3 bucket to store custom JSON. If you do not provide a value, the script creates a new S3 bucket.	No	string	`aws-apply-chef-application-manager-transition-data-`*`account-id`

Notes:

- If you use a private GitHub repository, you must create a new `Ed25519` host key for SSH. This is because GitHub changed which keys are supported in SSH and removed the unencrypted Git protocol. For more information about the `Ed25519` host key, see the GitHub blog post Improving Git protocol security on GitHub. After you generate a new `Ed25519` host key, create a Systems Manager `SecureString` parameter for the SSH key and use the `SecureString` parameter name as the value for the `--repo-private-key` parameter. For more information about how to create a Systems Manager `SecureString` parameter, see Create a SecureString parameter (AWS CLI) or Create a Systems Manager parameter (console) in the *AWS Systems Manager User Guide*.
- The `--http-username`, `--http-password` and `--repo-private-key` parameters refer to the name of a Systems Manager `SecureString` parameter. The migration script uses these parameters when you run the `AWS-ApplyChefRecipes` document.
- The `--http-username` parameter requires that you also specify a value for the `--http-password` parameter.
- The `--http-password` parameter requires that you also specify a value for the `--http-username` parameter.
- Do not set values for both `--http-password` and `--repo-private-key`. Provide either a Systems Manager `SecureString` parameter name of an SSH key (`--repo-private-key`), or a repository user name (`--http-username`) and password (`--http-password`).

Step 4: Run the script

When you run `python3 stack_exporter.py`, you can either provision the application, or create a starter template by setting the value of the `--provision-application` parameter to `FALSE`.

Example 1: Provision a Systems Manager Application Manager application

The following command gets information about an existing OpsWorks layer, and provisions an application using the newer OpsWorks architecture, which achieves a result similar to the Chef version configured for the stack. The script provisions all required resources, such as Auto Scaling groups by using CloudFormation, and then registers the application in Systems Manager Application Manager.

Replace *`stack-region`* and *`layer-id`* with the values for your OpsWorks stack and layer.

```
python3 stack_exporter.py \
     --layer-id layer-id \
     --region stack-region
```

Example 2: Generate a template

The following command gets information about an existing OpsWorks layer and generates a CloudFormation template. The template, if provisioned, achieves a result similar to using Chef 14. In this example, no resources are provisioned, because the `--provision-application` parameter is set to `FALSE`.

Replace *`stack-region`* and *`layer-id`* with the values for your OpsWorks stack and layer.

```
python3 stack_exporter.py \
    --layer-id layer-id \
    --region stack-region \
    --provision-application FALSE
```

After running the command, you can review the template in the Application Manager template library in Systems Manager, and you can also provision the template. For more information about viewing the template library, see Working with the template library in the *AWS Systems Manager User Guide*.

Step 5: Provision a CloudFormation stack

> **Note**
>
> You only need to complete this step if you set the `--provision-application` parameter for the script to FALSE.

When you specify the `--provision-application` parameter with a value of FALSE, the script output provides the name and URL for the CloudFormation template. This template represents a proposed replacement for your existing OpsWorks stack and layer.

You can provision the template by using the Application Manager template library (recommended), or by using CloudFormation. For more information about working with the template library, see Working with the template library in the *AWS Systems Manager User Guide*.

Step 6: Review the provisioned resources

You are now ready to review the provisioned resources.

1. Review resources for the provisioned stack using the AWS CloudFormation console.
   a. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation and choose **Stacks**.
   b. On the **Stacks** page, choose the stack, and then choose the **Resources** tab.
   c. On the **Resources** tab, review the listed resources for your stack. The list of resources includes an EC2 Auto Scaling group, which you can review in the Auto Scaling console, or AWS CLI.
2. Review the resources for the application using Systems Manager Application Manager.
   a. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager/.
   b. In the navigation pane, choose **Application Manager**.
   c. In the **Applications** section, choose the custom application. Application Manager opens the **Overview** tab.
   d. Choose the **Resources** tab. The **Resources** tab shows all resources that were migrated for your OpsWorks stack and layer. The application name includes the name of the OpsWorks stack, and is formatted as *app*-*stack-name*-*suffix* where *suffix* represents the first

six characters of the stack ID. For more information about viewing resources in Application Manager, see Viewing application resources in the *AWS Systems Manager User Guide*.

Step 7: Start an instance

After you've provisioned an instance, you are ready to test the instance. At this point, there are no instances running.

To take your instances online, adjust the `Min`, `Max`, and `Desired capacity` values for the Auto Scaling group to a number that makes sense for your application. Initially, you may want to set these values to 1, to bring a single instance online, and verify the instance performs all expected actions including running your custom Chef recipes.

Step 8: Review the instance

After you've started an instance, verify it runs as expected.

1. Review the Chef `startup` and `terminate` logs located in the S3 bucket specified by the script's `--command-logs-bucket` parameter. By default, the logs are stored in a bucket with the name `aws-opsworks-application-manager-logs-`*`account-id`*.

   a. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.

   b. Choose the bucket containing your logs.

   c. Navigate to the `ApplyChefRecipes` prefix to view your logs.

2. Check Application Load Balancer connectivity and health.

   Take the following steps to view the access logs for your load balancer. You can specify the S3 bucket where you want to store the load balancer access logs by using the script's `--lb-access-logs-path` parameter.

   a. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.

   b. Choose your S3 bucket, and then navigate to the prefix containing your logs.

3. Verify the instance passes all Auto Scaling and Application Load Balancer health checks (if you've configured any).

   You can view information about Auto Scaling health on the new **Instances** tab.

a. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager/.

b. In the navigation pane, choose **Application Manager**.

c. In the **Applications** section, choose **Custom applications**.

d. Choose the application in the list. Application Manager opens the **Overview** tab.

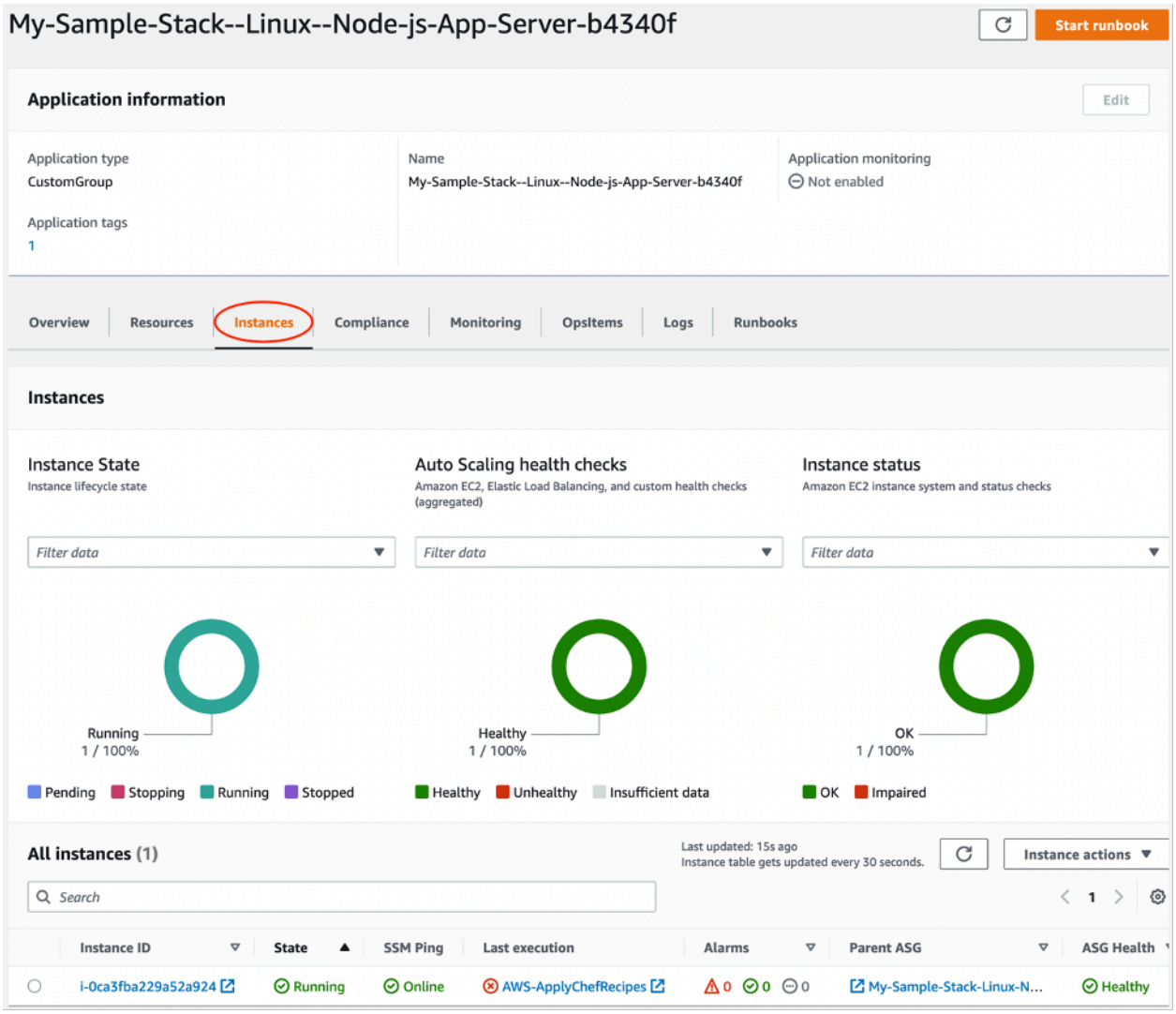
e. Choose the **Instances** tab to view information about Auto Scaling health.

After you verify that Chef recipes run successfully, you can decrease the Auto Scaling group capacity to terminate the instance. If you have any custom termination recipes, verify the recipes operate as expected.

Step 9: Monitor and run operations on your instances using Systems Manager Application Manager

You can now monitor and run operations on your instances using a new **Instances** tab on the Application Manager page. For more information about working with the **Instances** tab, see Working with your application instances in the *AWS Systems Manager User Guide*.

You can use the **Instances** tab to view multiple AWS instances in one place. Using this tab, you can view information about instance health and troubleshoot issues.



Take the following steps to view the **Instances** tab.

1. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. In the navigation pane, choose **Application Manager**.
3. In the **Applications** section, choose **Custom applications**.
4. Choose the application in the list. Application Manager opens the **Overview** tab.
5. Choose the **Instances** tab to view information about the status of your instance and EC2 health.

FAQ

The following FAQs provide answers to some common questions.

Topics

- Which AWS OpsWorks Stacks versions can I migrate?
- Which Chef versions can my migrated instances use?
- Which repository types can I migrate?
- Can I continue using a private Git repository?
- What SSH keys can I use to access my instances?
- Why are my instances automatically scaling in and out?
- Can I turn off Auto Scaling?
- Can I perform kernel and package updates on launched EC2 instances?
- Why don't the EBS volumes in my instances contain any data?
- Why aren't the EBS volumes described in my launch template mounted?
- Where can I find Chef recipe and Mount EBS volume logs?
- Where can I find the debug log for the migration script?
- Does the migration script support CloudFormation template versioning?
- Can I migrate multiple layers?
- How do I create a SecureString parameter?
- How can I protect instances in the new Auto Scaling group from termination events?
- What load balancers are available with the migration script?
- Are custom cookbook configure recipes migrated?
- Can I run deploy and undeploy recipes on my newly created instances?
- Can I change what subnets my Auto Scaling group spans?

Which AWS OpsWorks Stacks versions can I migrate?

You can only migrate Chef 11.10 and Chef 12, Amazon Linux, Amazon Linux 2, Ubuntu, and Red Hat Enterprise Linux 7 stacks.

Which Chef versions can my migrated instances use?

Migrated instances can use Chef versions 11 through 14.

> **Note**
>
> Windows stack migration is not supported.

Which repository types can I migrate?

You can migrate S3, Git, and HTTP repository types.

Can I continue using a private Git repository?

Yes, you can continue to use a private Git repository.

If you use a private GitHub repository, you must create a new `Ed25519` host key for SSH. This is because GitHub changed which keys are supported in SSH and removed the unencrypted Git protocol. For more information about the `Ed25519` host key, see the GitHub blog post Improving Git protocol security on GitHub. After you generate a new `Ed25519` host key, create a Systems Manager `SecureString` parameter for this SSH key and use the parameter name as the value for the `--repo-private-key` parameter. For more information about how to create a Systems Manager `SecureString` parameter, see Create a SecureString parameter (AWS CLI) in the *AWS Systems Manager User Guide*.

For any other Git repository type, create a Systems Manager `SecureString` parameter for this SSH key and use the parameter name as the value for the script's `--repo-private-key` parameter.

What SSH keys can I use to access my instances?

When you run the script, the script migrates the SSH keys and instances configured in the stack. You can use the SSH keys to access your instance. If SSH keys are provided for the stack and instance, the script uses the keys from the stack. If you are not sure which SSH keys to use, view the instances in the EC2 console (https://console.aws.amazon.com/ec2/). The **Details** page in the EC2 console shows the SSH keys for your instance.

Why are my instances automatically scaling in and out?

Auto Scaling scales instances based on the scaling rules for the Auto Scaling group. You can set the **Min**, **Max**, and **Desired capacity** values for your group. The Auto Scaling group automatically scales your capacity accordingly when you update these values.

Can I turn off Auto Scaling?

You can turn off Auto Scaling by setting the Auto Scaling group's **Min**, **Max**, and **Desired capacity** values to the same number. For example, if you want to always have ten instances, set the **Min**, **Max**, and **Desired capacity** values to ten.

Can I perform kernel and package updates on launched EC2 instances?

By default, kernel and packages updates occur when the EC2 instance boots. Use the following steps to perform kernel or package updates on a launched EC2 instance. For example, you may want to apply updates after running deploy or configure recipes.

1. Connect to your EC2 instance.
2. Create the following `perform_upgrade` function and run it on your instance.

```
perform_upgrade() {
    #!/bin/bash
    if [ -e '/etc/system-release' ] || [ -e '/etc/redhat-release' ]; then
     sudo yum -y update
    elif [ -e '/etc/debian_version' ]; then
     sudo apt-get update
     sudo apt-get dist-upgrade -y
    fi
}
perform_upgrade
```

3. After the kernel and package updates, you may need to reboot your EC2 instance. To check whether you a reboot is required, create the following `reboot_if_required` function and run it on your EC2 instance.

```
reboot_if_required () {
 #!/bin/bash
 if [ -e '/etc/debian_version' ]; then
   if [ -f /var/run/reboot-required ]; then
     echo "reboot is required"
   else
     echo "reboot is not required"
   fi
 elif [ -e '/etc/system-release' ] || [ -e '/etc/redhat-release' ]; then
  export LC_CTYPE=en_US.UTF-8
  export LC_ALL=en_US.UTF-8
```

```
  LATEST_INSTALLED_KERNEL=`rpm -q --last kernel | perl -X -pe 's/^kernel-(\S+).*/
$1/' | head -1`
  CURRENTLY_USED_KERNEL=`uname -r`
  if [ "${LATEST_INSTALLED_KERNEL}" != "${CURRENTLY_USED_KERNEL}" ];then
     echo "reboot is required"
  else
     echo "reboot is not required"
  fi
 fi
}
reboot_if_required
```

4. If running the `reboot_if_required` results in a `reboot is required` message, reboot the EC2 instance. If you receive a `reboot is not required` message, you do not need to reboot the EC2 instance.

Why don't the EBS volumes in my instances contain any data?

When you run the script, the script migrates the configuration of the EBS volumes, creating a replacement architecture for your OpsWorks stack and layer. The script does not migrate actual instances or the data contained in the instances. The script only migrates the configuration of EBS volumes at the layer level and attaches the empty EBS volumes to launched EC2 instances.

Take the following steps to pull data from your previous instances' EBS volumes.

1. Take a snapshot of your previous instances EBS volumes. For more information about creating a snapshot, see Create Amazon EBS snapshot in the *Amazon EC2 User Guide*.
2. Create a volume from your snapshot. For more information about creating a volume from a snapshot, see Create a volume from a snapshot in the *Amazon EC2 User Guide*.
3. Attach the volume you created to the instances. For more information about attaching volumes, see Attach an Amazon EBS volume to an instance in the *Amazon EC2 User Guide*.

Why aren't the EBS volumes described in my launch template mounted?

If you provide a launch template ID for the `--launch-template` parameter with EBS volumes, the script attaches the EBS volumes, but does not mount the volumes. You can mount the attached EBS volumes by running the `MountEBSVolumes` RunCommand document that the script created for the launched EC2 instance.

If you don't set `--launch-template` parameter, the script creates a template, and when the Auto Scaling group launches a new EC2 instance, the Auto Scaling group automatically attaches the EBS volumes and then runs the `SetupAutomation` command to mount the attached volumes to the mount points configured in the layer settings.

Where can I find Chef recipe and Mount EBS volume logs?

OpsWorks delivers the logs to an S3 bucket that you can specify by providing a value for the `--command-logs-bucket` parameter. The default S3 bucket name has the format: `aws-opsworks-stacks-application-manager-logs-`*`account-id`*. Chef recipe logs are stored in the `ApplyChefRecipes` prefix. Mount EBS volume logs are stored in the `MountEBSVolumes` prefix. All layers that are migrated from a stack deliver logs to the same S3 bucket.

> **Note**
>
> - The S3 bucket's Lifecycle configuration includes a rule to delete the logs after 30 days. If you want to keep the logs for more than 30 days, you must update the rule in the S3 bucket's Lifecycle configuration.
> - Currently, OpsWorks only logs Chef `setup` and `terminate` recipes.

Where can I find the debug log for the migration script?

The script places debug logs in a bucket named `aws-opsworks-stacks-transition-logs-`*`account-id`*. You can find the debug logs in the `migration_script` folder of the S3 bucket under folders that match the name of the layer you migrated.

Does the migration script support CloudFormation template versioning?

The script generates Systems Manager documents of type CloudFormation that create a replacement for the layer or stack you want to migrate. Running the script again, even with the same parameters, exports a new version of the previously-exported layer template. The template versions are stored in the same S3 bucket as the script logs.

Can I migrate multiple layers?

The script's `--layer-id` parameter passes in a single layer. To migrate multiple layers, rerun the script and pass in a different `--layer-id`.

Layers that are part of the same OpsWorks stack are listed under the same application in Application Manager.

1. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. In the navigation pane, choose **Application Manager**.
3. In the **Applications** section, choose **Custom applications**.
4. Choose your application. The application name begins with app-*stack-name*-*first-six-characters-stack-id*.
5. The top level element starting with app, shows all components which correspond to your OpsWorks stack. This includes components corresponding to your OpsWorks layer.
6. Choose the component corresponding to the layer to view the resources for the layer. The components representing OpsWorks layers are also visible from the **Custom applications** section as individual applications.

How do I create a `SecureString` parameter?

You can use Systems Manager to create a `SecureString` parameter. For more information about how to create a Systems Manager `SecureString` parameter, see Create a SecureString parameter (AWS CLI) or Create a Systems Manager parameter (console) in the *AWS Systems Manager User Guide*.

You must provide a `SecureString` parameter as the value for the `--http-username`, `--http-password`, or `--repo-private-key` parameters.

How can I protect instances in the new Auto Scaling group from termination events?

You can protect instances by setting the `--enable-instance-protection` parameter to `TRUE` and adding a `protected_instance` tag key to each EC2 instance you want to protect from termination events. When you set the `--enable-instance-protection` parameter to `TRUE` and add a `protected_instance` tag key, the script adds a custom termination policy to your new Auto Scaling group and suspends the `ReplaceUnhealthy` process. Instances with the `protected_instance` tag key are protected from the following termination events:

- Scale in events
- Instance refresh
- Rebalancing

- Instance max lifetime
- Allow listing instance termination
- Termination and replacement of unhealthy instances

> **Note**
>
> You must set the `protected_instance` tag key on instances you want to protect. The tag key is case sensitive. Any instance with that tag key is protected regardless of the tag value. To reduce the run time of the custom termination policy, you can increase the default number of instances the Lambda function uses to filter for protected instances by updating the value for the `default_sample_size` function code variable. The default value is 15. If you increase the `default_sample_size`, you may need to increase the memory allocated to the Lambda function, which would increase the cost of your Lambda function. For information about AWS Lambda pricing, see AWS Lambda Pricing.

What load balancers are available with the migration script?

The script provides three load balancer options.

- (Recommended) Create a new Application Load Balancer. By default, the script creates a new Application Load Balancer. You can also set the `--lb-type` parameter to `ALB`. For more information about Application Load Balancers, see What is an Application Load Balancer? in the *Elastic Load Balancing User Guide*.
- If an Application Load Balancer is not an option, create a Classic Load Balancer by setting the `--lb-type` parameter to `Classic`. If you select this option, your existing Classic Load Balancer attached to your OpsWorks layer is kept separate from your application. For more information about Application Load Balancers, see What is a Classic Load Balancer? in the *Elastic Load Balancing: Classic Load Balancers User Guide*.
- You can attach an existing load balancer by setting the `--lb-type` parameter to `None`.

> **Important**
>
> We recommend creating new Elastic Load Balancing load balancers for your AWS OpsWorks Stacks layers. If you choose to use an existing Elastic Load Balancing load balancer, you should first confirm that it is not being used for other purposes and

has no attached instances. After the load balancer is attached to the layer, OpsWorks removes any existing instances and configures the load balancer to handle only the layer's instances. Although it is technically possible to use the Elastic Load Balancing console or API to modify a load balancer's configuration after attaching it to a layer, you should not do so; the changes will not be permanent.

To attach your existing OpsWorks layer load balancer to your Auto Scaling group

1. Run the migration script with the `--lb-type` parameter set to `None`. When the value is set to `None`, the script does not clone or create a load balancer.
2. After the script deploys the CloudFormation stack, update the Auto Scaling groups `Min` `Max` and `Desired capacity` values, and then test your application.
3. Choose `Link to the template` shown in the script's output. If you closed your terminal, take these steps to access the template.
   a. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager/.
   b. In the navigation pane, choose **Application Manager**.
   c. Choose **CloudFormation stacks** and then choose **Template library**.
   d. Choose **Owned by me** and locate your template.
4. From the CloudFormation template, choose **Edit** from the **Actions** menu.
5. Update the `LabelBalancerNames` property within the `ApplicationAsg` resource section of the CloudFormation template.

```
ApplicationAsg:
   DependsOn: CustomTerminationLambdaPermission
   Properties:
   #(other properties in ApplicationAsg to remain unchanged)
      LoadBalancerNames:
        - load-balancer-name
      HealthCheckType: ELB
```

6. If you want your Auto Scaling group instances health check to also use the load balancer's health check, remove the section below `HealthCheckType` and enter `ELB`. If you only require EC2 health checks, you do not need to change the template.

7. Save your changes. Saving creates a new default version of the template. If this is the first time you've run the script for the layer and the first time you've saved changes in the console, the newer version is 2.
8. From **Actions** , choose **Provision stack**.
9. Confirm that you want to use the default version of the template. Be sure **Select an existing stack** is selected and choose the CloudFormation stack to update.
10. Choose **Next** for each of the subsequent pages until you see the **Review and Provision** page. On the **Review and Provision** page, choose both **I acknowledge that AWS CloudFormation might create IAM resources with custom names** and **I understand that changes in the selected template can cause AWS CloudFormation to update or remove existing AWS resources.**
11. Choose **Provision stack**.

If you need to roll back your updates, take the following steps.

1. Choose **Actions** and then choose **Provision stack**.
2. Choose **Pick one of the existing versions** and then choose the previous template version.
3. Choose **Select an existing stack** and then choose the CloudFormation stack to update.

Are custom cookbook configure recipes migrated?

Configure custom cookbooks are not supported to run during a setup event. The script migrates custom cookbook configure recipes and creates a Systems Manager Automation runbook for you. However, you must run the recipes manually.

Take the following steps to run your configure recipes.

1. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. In the navigation pane, choose **Application Manager**.
3. In the **Applications** section, choose **Custom applications**.
4. Choose your application. The application name begins with `app-`*`stack-name`*.
5. Choose **Resources** and then choose the configure runbook.
6. Choose **Execute Automation**.
7. Choose the instance IDs for which you want to run the configure recipes and then choose **Execute**.

Can I run deploy and undeploy recipes on my newly created instances?

The script can create three possible Automation runbooks depending on the configuration of your layer.

- Setup
- Configure
- Terminate

The script can also create the following Systems Manager parameters that contain input values for the `AWS-ApplyChefRecipes Run Command` document.

- Setup
- Deploy
- Configure
- Undeploy
- Terminate

When a scale-out event happens, the setup Automation runbook runs automatically. This includes the setup and deploy custom cookbook recipes from your original OpsWorks layer. When a scale-in event happens, the terminate Automation runbook runs automatically. The terminate Automation runbook contains the shutdown recipes from your original OpsWorks layer.

If you want to run undeploy or configure recipes manually, take the following steps.

1. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager/.
2. In the navigation pane, choose **Application Manager**.
3. In the **Applications** section, choose **Custom applications**.
4. Choose your application. The application name begins with app-*stack-name*-*first-six-characters-stack-id*. Application Manager opens the **Overview** tab.
5. Choose **Resources** and then choose the configure Automation runbook.
6. Choose **Execute Automation**.
7. For the `applyChefRecipesPropertiesParameter` Automation runbook input parameter, reference the correct Systems Manager parameter. The Systems Manager parameter name follows the format `/ApplyChefRecipes-Preset/`*OpsWorks-stack-name*-*OpsWorks-*

layer-name-*first-six-characters-stack-id*/*event* , where the value for *event* is `Configure`, `Deploy`, or `Undeploy` depending on the recipes you want to run.

8. Choose the instance IDs where you want to run the recipes and choose **Execute**.

Can I change what subnets my Auto Scaling group spans?

By default, the Auto Scaling group spans all subnets in your OpsWorks stack VPC. To update which subnets to span, take the following steps.

1. Choose `Link to the template` shown in the script's output. If you closed your terminal, take these steps to access the template.

   a. Open the Systems Manager console at https://console.aws.amazon.com/systems-manager/.

   b. In the navigation pane, choose **Application Manager**.

   c. Choose **CloudFormation stacks** and then choose **Template library**.

   d. Choose **Owned by me** and locate your template.

2. From **Actions**, choose **Provision stack**.

3. Confirm that you want to use the default template. Choose **Select an existing stack** and then choose the CloudFormation stack to update.

   > **Note**
   >
   > If you ran the script with the `--provision-application` parameter set to `FALSE`, you must create a new CloudFormation stack.

4. For the `SubnetIDs` parameter, provide a comma separated list of the subnet IDs that you want your Auto Scaling group to span.

5. Choose **Next** until you see the **Review and Provision** page.

6. On the **Review and Provision** page, choose **I acknowledge that AWS CloudFormation might create IAM resources with custom names** and **I understand that changes in the selected template can cause AWS CloudFormation to update or remove existing AWS resources**.

7. Choose **Provision stack**.

Troubleshooting

This section contains some common issues, and suggested solutions for those issues.

Topics

- Provided principal is not valid
- Unable to delete CloudFormation stack when Auto Scaling group protected instances are enabled
- Access denied error when providing existing S3 bucket and prefix

Provided principal is not valid

Problem: You receive an error message stating that the principal you provided is not valid.

Cause: This occurs because the Auto Scaling group doesn't have a service role.

Solution: Create an Auto Scaling group in the Region where the error occurred. Creating an Auto Scaling group creates the necessary service-linked role for your custom termination policy.

Unable to delete CloudFormation stack when Auto Scaling group protected instances are enabled

Problem: The `--enable-instance-protection` parameter is set to TRUE and some of your Auto Scaling group's EC2 instances are protected with the `protected_instance` tag key, which prevents your AWS CloudFormation stack from being completely deleted.

Cause: The EC2 instances have a `protected_instance` tag key which protects them from termination events.

Solution: Remove the `protected_instance` tag key from the EC2 instances. This allows the Auto Scaling group to scale down. After the Auto Scaling group scales down, you can delete the AWS CloudFormation stack.

Access denied error when providing existing S3 bucket and prefix

Problem: You receive an `AccessDenied` error when you provide an existing S3 bucket and prefix.

Cause: The S3 bucket policy does not provide the necessary permissions to deliver the load balancer logs to the bucket.

Solution: Update the S3 bucket policy to allow the script to deliver the load balancer access logs to the bucket. For more information about how to update the bucket policy, see Enable access logs for your Application Load Balancer in the *Elastic Load Balancing: Application Load Balancers User Guide*.

Using the AWS OpsWorks Stacks Detach in Place tool

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible.

This section describes how to use the AWS OpsWorks Stacks Detach in Place tool to detach your OpsWorks instances from the OpsWorks Stacks service.

The instances you detach will remain in your AWS account, but you'll no longer be able to manage them using OpsWorks. Instead, you'll use Amazon EC2, AWS Systems Manager, or any EC2 compatible approach to configure and manage the instances.

At a high level, the detachment process involves the following steps:

1. The tool performs validation checks to ensure the resources are ready for detachment.
2. The tool exports the Custom JSON from your OpsWorks stack and stores it as an object in Amazon S3.
3. The tool creates Systems Manager Automation documents representing each OpsWorks Stacks lifecycle event.
4. The tool creates an AWS Service Catalog AppRegistry Catalog for all instances that are being detached and detaches any Elastic Load Balancing (ELB) load balancers from the OpsWorks layers.
5. Finally, the tool detaches and deregisters other resources including Amazon Relational Database Service (Amazon RDS) instances.

How the process works

The Detach In Place tool provides the following 3 commands and a wizard like experience that guides you through a series of steps to check and configure your instances before proceeding to detach your layer.

Command	Description
`handle-prerequisites`	This command analyses whether all instances in a layer are eligible for detachment and resolves prerequisites. The instances must be in a healthy state in OpsWorks, they cannot have time or load based auto scalers, and must have the latest OpsWorks Agent version installed. Additionally, the command checks if all instances have the permissions required to support the SSM Agent and if the latest SSM Agent version is installed. The command will install the SSM Agent if it is not present and will update the SSM Agent if it's not using the latest version. The command will also add any necessary permissions.
`detach`	This command detaches all OpsWorks instances for the specified layer. First, the command will run a prerequis ites check to ensure the layer qualifies for detachment. If you do not want to resolve the prerequisites, you're given the option to force detach. Next, the command will indicate that all tags added to your instances through OpsWorks tagging APIs or through propagation of tags from your layers and stacks, will be retained.

Command	Description
	You can remove any of these tags using relevant EC2 APIs after the detachment is complete. Then, the command will check if you want to export the Chef related configuration to SSM parameters. If you have a Classic Load Balancer attached to the layer, the command will ask if it can detach the load balancer in order to prevent any downtime.
`cleanup`	This command deletes all entities in OpsWorks from your account. It will terminate the instances and delete all stacks. This should be used for resources that are no longer needed as a last step to clean up the account. **Note** We recommend you run the new setup for a few days prior to running the `cleanup` command. This ensures that any necessary configurations from the stack are readily available if needed.

Limitations

The main purpose of the Detach In Place tool is to safely detach the OpsWorks Stacks instances. This section summarises the limitations of the tool.

- **Windows SSM Agent** – If the SSM Agent is not installed on the instance, you'll need to manually install it. The same applies if the Agent is not updated to the latest version.

- **Time/Load Auto Scaling instances** – The detachment tool doesn't support instances with Auto Scaling enabled. You must disable Auto Scaling on instances you want to detach.
- **Permissions** – The detachment tool doesn't create or generate IAM entities specified on the **Permissions** page of the OpsWorks console.
- **Apps** – The detachment tool doesn't create or generate apps outside of OpsWorks.

Getting started

Step 1: Verify the prerequisites are met

All 3 commands of the Detach In Place tool are Python scripts, which you can run locally, on EC2 instance, or by using AWS CloudShell.

AWS CloudShell is a browser-based shell that gives you command-line access to the AWS resources in the selected AWS Region. AWS CloudShell comes pre-installed with popular tools (such as AWS CLI and Python). When using AWS CloudShell, you use the same credentials that you use to sign in to the console.

This walkthrough assumes that you're using AWS CloudShell.

Step 2: Download the script

1. Download the zip file that contains the migration script and all the relevant files by running the following command:

   ```
   aws s3api get-object \
   --bucket detach-in-place-bucket-prod-us-east-1 \
   --key detach_in_place_script.zip detach_in_place_script.zip
   ```

2. Unzip the file by running the following command.

   ```
   unzip detach_in_place_script.zip
   ```

   After the file is unzipped, the following files are available:

   - README.md
   - LICENSE
   - NOTICE

- requirements.txt
- TODO.py

3. If necessary, install `pipenv` by running the following command.

```
pip install pipenv
```

Step 3: Run the script

First, set up your environment so that you can run the script by running the following commands.

```
pipenv install -r requirements.txt
pipenv shell
```

Then, review the script parameters.

Command	Parameter	Description	Type	Required	Default
`handle-prerequisites`	`--layer-id`	The ID of the layer you want to detach.	String	Yes	-
	`--region`	The Region of the OpsWorks stack. If your OpsWorks stack Region and API endpoint Region are different, use the stack Region. This is the same Region as the other resources part of your OpsWorks stack (for example, EC2 instances and subnets).	String	No	us-east-1
`detach`	`--layer-id`	ID of the layer you want to detach.	String	Yes	-

Command	Parameter	Description	Type	Required	Default
	`--batch-size`	Number of instances to detach from a layer (for example, 5).	String	No	-
	`--region`	The Region of the OpsWorks stack. If your OpsWorks stack Region and API endpoint Region are different, use the stack Region. This is the same Region as the other resources part of your OpsWorks stack (for example, EC2 instances and subnets).	String	No	us-east-1
`cleanup`	`--stack-id`	ID of the stack you want to delete.	String	No	Mutually exclusive, you must specify either a layer ID or a stack ID
	`--layer-id`	ID of the layer you want to delete	String	No	

Command	Parameter	Description	Type	Required	Default
	`--region`	The Region of the OpsWorks stack. If your OpsWorks stack Region and API endpoint Region are different, use the stack Region. This is the same Region as the other resources part of your OpsWorks stack (for example, EC2 instances and subnets).	String	No	us-east-1

You can see the available options for the `detach`, `handle-prerequisites` and `cleanup` commands by running the commands with the `--help` option as follows:

```
python3 layer_detacher.py detach --help
python3 layer_detacher.py handle-prerequisites --help
python3 layer_detacher.py cleanup --help
```

You're now ready to get started. The following examples show how you can run the commands for different use cases.

Examples:

- Example 1: Check if a layer fulfills all prerequisites and is eligible for detachment
- Example 2: Detach all instances of a layer
- Example 3: Detach all instances of a layer in batches
- Example 4: Clean up all resources for a layer and delete the layer
- Example 5: Clean up all resources for a stack and delete the stack

Example 1: Check if a layer fulfills all prerequisites and is eligible for detachment

The following command reads information about an OpsWorks layer (and the instances it includes) and checks if the following prerequisites are met:

- All instances are online.
- There are no Load/Time Auto Scaling instances.
- All instances have the latest OpsWorks Agent.
- All instances have the latest SSM Agent installed and configured.
- All instances have an SSH key pair.
- Every instance belongs to exactly one layer.

```
python3 layer_detacher.py handle-prerequisites \
--layer-id opsworks-layer-id \
--region opsworks-stack-region
```

Example 2: Detach all instances of a layer

The following command will iterate over all instances of the layer, check if the instances meet the prerequisites and try to detach in parallel all instances that meet the prerequisites. If one or more prerequisites are not met, the command will provide a force detach option for the remaining non-compliant instances.

Before detaching any instance the command will:

1. Save the Custom JSON and upload it to S3.
2. Create SSM Automation documents for every OpsWorks lifecycle event for the layer and upload the execution logs for the Automation documents to S3.
3. Create an AppRegistry application for all instances that will be detached. The application has a Resource Group associated with it which holds all detached instances and resources. The resources include SSM Automation documents and SSM parameters that hold information about lifecycle events and custom Chef recipes.
4. Detaches the Classic Load Balancer from the layer, if one exists.

This command will modify only OpsWorks resources. The status of EC2 instances will remain the same.

```
python3 layer_detacher.py detach \
--layer-id opsworks-layer-id \
--region opsworks-stack-region
```

Example 3: Detach all instances of a layer in batches

The following command does the same as the previous example. The only difference is that it detaches the instances in batches.

This command will modify only OpsWorks resources. The status of EC2 instances will remain the same.

```
python3 layer_detacher.py detach \
--layer-id opsworks-layer-id \
--region opsworks-stack-region \
--batch-size 5
```

Example 4: Clean up all resources for a layer and delete the layer

The following command will iterate over all resources for a layer and delete them. In more detail, it will stop and delete all instances in OpsWorks and EC2, detach the load balancer and deregister Amazon RDS instances, elastic IPs and volumes. After cleaning up the resources, it will delete the layer.

This command will delete OpsWorks resources and EC2 instances. If you want your EC2 instances to remain untouched, use the `detach` command before using the `cleanup` command. That way the `cleanup` command will delete all the remaining resources.

```
python3 layer_detacher.py cleanup \
--layer-id opsworks-layer-id \
--region opsworks-stack-region
```

Example 5: Clean up all resources for a stack and delete the stack

The following command will iterate over all layers and then iterate over the resources of each layer. For each layer, the command will stop and delete all instances in OpsWorks and EC2, detach load balancers, and deregister Amazon RDS instances, elastic IPs and volumes. Then, the command will delete the layer. The same process will be performed in every layer that belongs to this stack. Finally, after all layers are deleted, the stack will be removed.

This command will delete OpsWorks resources and EC2 instances. If you want your EC2 instances to remain untouched, use the `detach` command before using the `cleanup` command. That way the `cleanup` command will delete all the remaining resources.

```
python3 layer_detacher.py cleanup \
--stack-id opsworks-stack-id \
--region opsworks-stack-region
```

Step 4: Continue to operate your resources after detaching from OpsWorks

After running the `detach` command, the tool creates a new AWS Service Catalog AppRegistry application corresponding to the detached layer. The application name follows the format *layer-name*`---`*layer-id*. It also adds the `OpsWorksLayerId` tag to uniquely identify the application matching the detached layer.

To add new AWS resources to this application (for example, new EC2 instances), you can do one of the following:

1. Tag the resource with the unique application tag of the AppRegistry application:

   Tag key: `awsApplication`

   Value: `arn:aws:resource-groups:`*region*`:`*account-id*`:group/`*application-name*`/`*application-id*`>`
2. Run the associate-resource command.

Additionally, for each AppRegistry application, a Resource Group is created. The Resource Group contains the following tags.

Tag key	Value
`EnableAWSServiceCatalogAppRegistry`	`TRUE`
`aws:servicecatalog:applicationName`	*application-name*
`aws:servicecatalog:applicationId`	*application-id*
`aws:servicecatalog:applicationArn`	`arn:aws:servicecatalog:` *region*`:`*account-id* `:/applications/` *application-id*

Performing tasks after detachment

The following table provides information about how to perform tasks after detachment:

Task	Description
Executing lifecycle events	After running the `detach` command and if you selected the option, the script creates 5 Automation documents matching the 5 OpsWorks lifecycle events. The name of each Automation document follows this format: *`layer-id`*`_`*`lifecycle-event`*`_automation_document` . To simulate OpsWorks behavior in Systems Manager, you'll need to manually trigger Automation executions when provisioning, terminating EC2 instances, or deploying/removing recipes.
Updating Custom JSON	Custom JSON for the stack and layer are stored in an S3 bucket specified during detachment, or alternatively in a new S3 bucket that gets created. The file names stored for the JSON files are the following: • `layercustomjson.json` • `stackcustomjson.json`
Changing your run list for lifecycle events	The run list for each lifecycle event is defined in the corresponding Automation document. To change the run list, find the Automation documents in the AppRegistry application and modify the `RunList` parameter.

Task	Description
	The process to update recipes and cookbooks is unchanged because `AWS-ApplyChefRecipes`, which the Automation documents trigger, supports the same source as OpsWorks.
Managing auto healing / auto scaling	When you detach an instance, the OpsWorks Agent uninstalls. Without the agent, OpsWorks cannot automatically heal or replace unhealthy instances, nor can it auto scale your fleet. To continue auto scaling and replacing failed instances, create an Amazon EC2 Auto Scaling group. The group will launch new instances to maintain its desired capacity when Amazon EC2 detects unhealthy instances that need replacing.
Managing Load Balancer	If your layer uses a Classic Load Balancer, the `detach` command will detach it prior to deregistering the instances. This is done to ensure that all ELB instance associations remain preserved on Amazon EC2 throughout the course of detachment, thus leading to zero downtime. After the process is completed you will be able to manage your ELB on EC2.

Task	Description
Connecting to your instances	When you run the `handle-prerequisites` or `detach` command, two checks occur: • The version of the SSM Agent and permissions • SSH keys The commands also offer you the option to update the SSM Agent and add required permissions so you can connect to instances using Session Manager. If SSH keys exist, you also have the option to SSH into the instance.

Using the Systems Manager Application Manager Instances tab

After detachment, you'll be able to view and manage your instances on the Application Manager Instances tab.

The **Instances** tab provides aggregate information about an application's EC2 instances, such as their status, health state, and last command status. Using this tab, you can view detailed information about individual instances such as command history, alarm states, Systems Manager agent health, and more. The **Instances** tab also provides a variety of actions, such as the ability to apply Chef recipes, start or stop an instance, or add or remove an instance from an Auto Scaling group.

Getting Started with OpsWorks Stacks

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks provides a rich set of customizable components that you can mix and match to create a stack that satisfies your specific purposes. The challenge for new users is understanding how to assemble these components into a working stack and manage it effectively. Here's how you can get started.

If you want to...	Complete this walkthrough:
Create a sample stack as quick as possible	Getting Started: Sample
Experiment with a Linux-based stack	Getting Started: Linux
Experiment with a Windows-based stack	Getting Started: Windows
Learn how to create your own Chef cookbooks	Getting Started: Cookbooks

If you have existing computing resources—Amazon EC2 instances or even *on-premises* instances that are running on your own hardware—you can incorporate them into a stack, along with instances that you created with OpsWorks Stacks. You can then use OpsWorks Stacks to manage all related instance as a group, regardless of how they were created.

Region Support

You can access OpsWorks Stacks globally; you can also create and manage instances globally. Users can configure OpsWorks Stacks instances to be launched in any AWS region except AWS GovCloud (US-West) and the China (Beijing) Region. To work with OpsWorks Stacks, instances must be able to connect to one of the following OpsWorks Stacks instance service API endpoints.

Resources can be managed only in the region in which they are created. Resources that are created in one regional endpoint are not available, nor can they be cloned to, another regional endpoint. You can launch instances in any of the following regions.

- US East (Ohio) Region
- US East (N. Virginia) Region
- US West (Oregon) Region
- US West (N. California) Region
- Canada (Central) Region (API only, not available for stacks created in the AWS Management Console.)

- Asia Pacific (Mumbai) Region
- Asia Pacific (Singapore) Region
- Asia Pacific (Sydney) Region
- Asia Pacific (Tokyo) Region
- Asia Pacific (Seoul) Region
- Europe (Frankfurt) Region
- Europe (Ireland) Region
- Europe (London) Region
- Europe (Paris) Region
- South America (São Paulo) Region

Getting Started with a Sample Stack

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This walkthrough shows how to use OpsWorks Stacks to quickly create a sample Node.js application environment with just a few mouse clicks and without writing code. When you are done, you have an Amazon Elastic Compute Cloud (Amazon EC2) instance running Chef 12, a Node.js HTTP server, and a web app that you can use to interact with Twitter and leave comments on a web page.

Note

Because completing this walkthrough automatically creates an instance with a type of c3.large, you cannot use this walkthrough, or the **Sample Stack** creation tool in OpsWorks Stacks, in the AWS Free Tier. Although using the **Sample Stack** creation tool in a VPC creates a t2.medium instance, VPCs are not currently available in the AWS Free Tier.

Step 1: Complete the Prerequisites

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You must complete the following setup steps before you can start the walkthrough. These setup steps include signing up for an AWS account, creating an administrative user, and assigning access permissions to OpsWorks Stacks.

Topics

- Sign up for an AWS account
- Create a user with administrative access
- Assign Service Access Permissions

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Assign Service Access Permissions

Enable access to the OpsWorks Stacks service (and related services that OpsWorks Stacks relies on) by adding the `AWSOpsWorks_FullAccess` and `AmazonS3FullAccess` permissions to your role or user.

For more information about adding permissions, see Adding IAM identity permissions (console).

You have now completed all of the setup steps and can start this walkthrough.

Step 2: Create a Stack

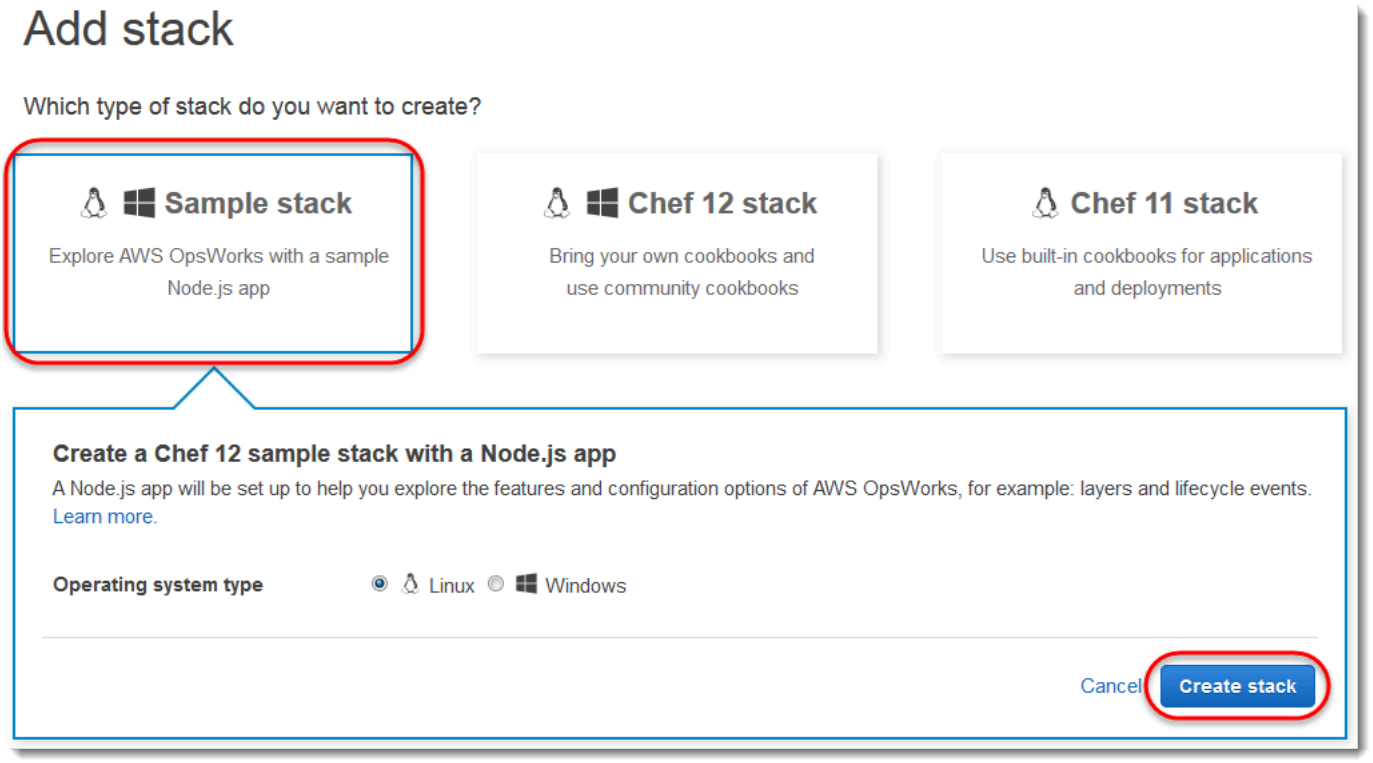
> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

In this step, you use the OpsWorks Stacks console to create a stack. A *stack* is a collection of instances (such as Amazon EC2 instances) and related AWS resources that have a common purpose and that you want to manage together. (For more information, see Stacks.) There will be only one instance for this walkthrough.

Before you begin this step, complete the prerequisites.

To create the stack

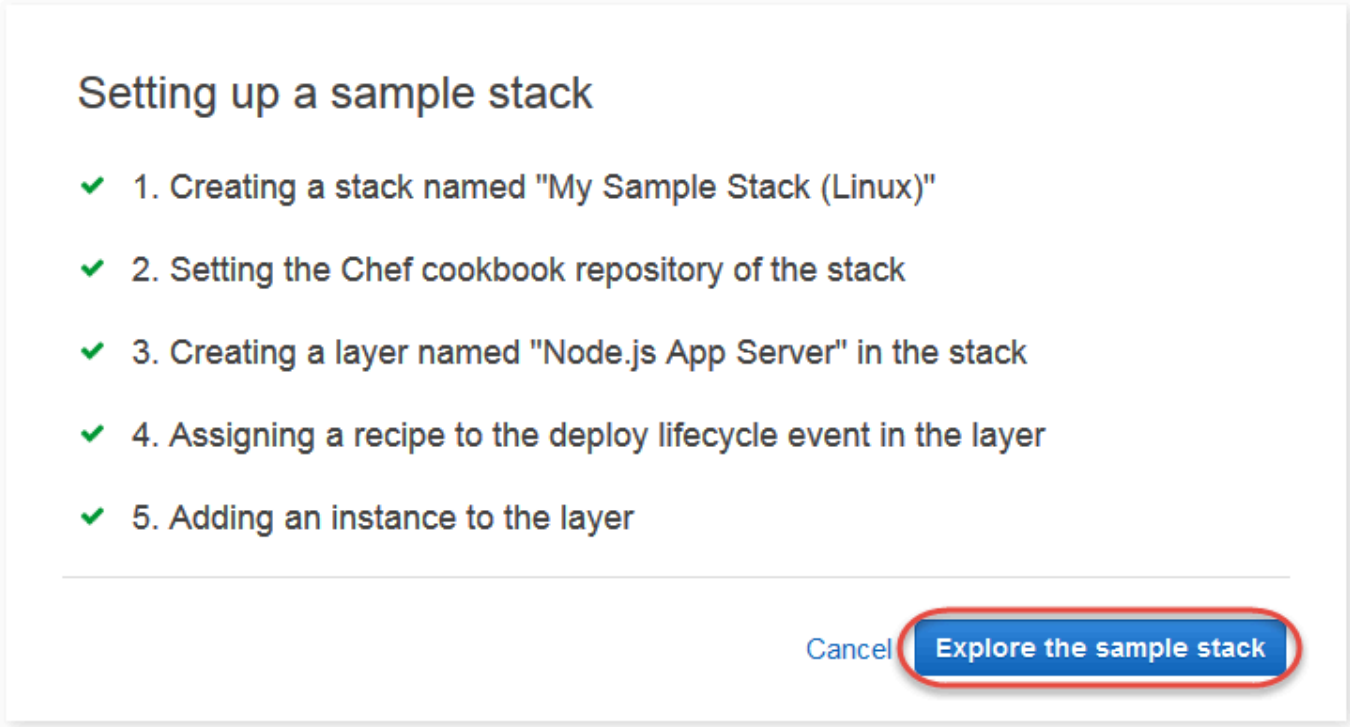
1. Sign in to the AWS Management Console and open the OpsWorks console at https://console.aws.amazon.com/opsworks/.
2. Do any of the following, if they apply:
   - If the **Welcome to OpsWorks Stacks** page is displayed, choose **Add your first stack** or **Add your first OpsWorks Stacks stack** (both choices do the same thing). The **Add stack** page displays.
   - If the **OpsWorks Dashboard** page is displayed, choose **Add stack**. The **Add stack** page displays.
3. With the **Add stack** page displayed, choose **Sample stack**, if it is not already chosen for you.
4. With **Linux** already chosen for **Operating system type**, choose **Create stack**:



5. OpsWorks Stacks creates a stack named **My Sample Stack (Linux)**. OpsWorks Stacks also adds all of the necessary components to deploy the app to the stack:

- A *layer*, which is a blueprint for a set of instances. It specifies things like the instance's settings, resources, installed packages, and security groups. (For more information, see Layers.) The layer is named **Node.js App Server**.
- An *instance*, which in this case is an Amazon Linux 2 EC2 instance. (For more information about instances, see Instances.) The instance's hostname is **nodejs-server1**.
- An *app*, which is code to run on the instance. (For more information about apps, see Apps.) The app is named **Node.js Sample App**.

6. After OpsWorks Stacks creates the stack, choose **Explore the sample stack** to display the **My Sample Stack (Linux)** page (if you complete this walkthrough multiple times, **My Sample Stack (Linux)** may have a sequential number after it, such as **2** or **3**):



In the next step, you will start the instance and deploy the app to the instance.

Step 3: Start the Instance and Deploy the App

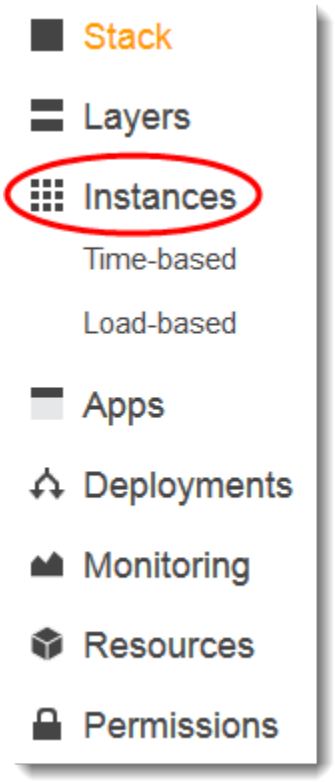
> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Now that you have an instance and an app, start the instance and deploy the app to the instance.

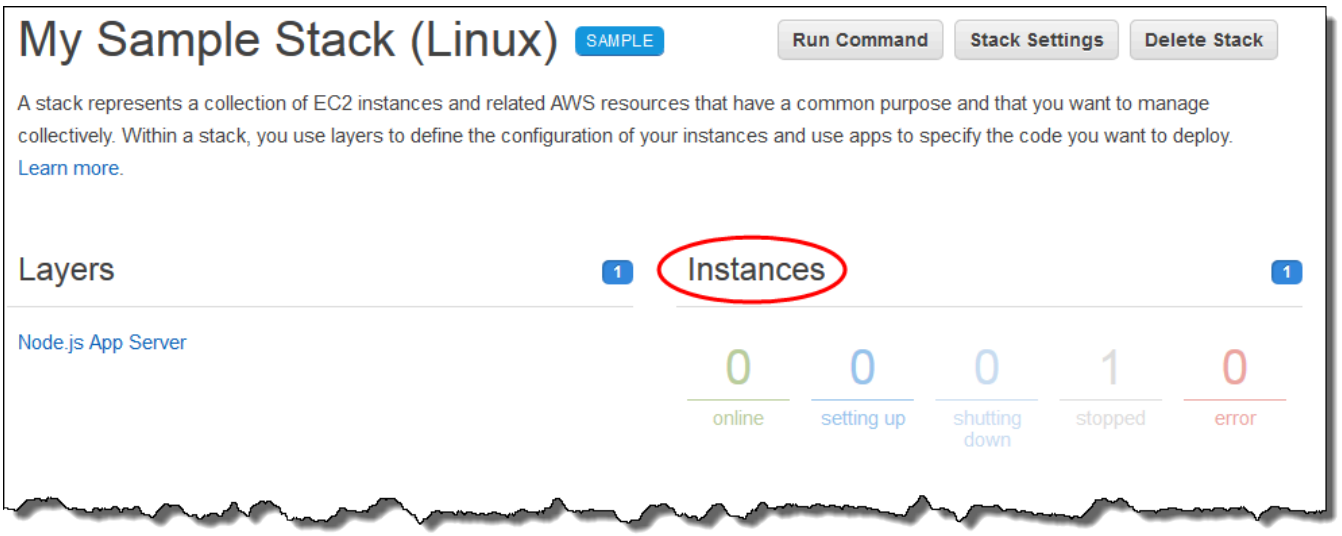
To start the instance and deploy the app

1. Do one of the following:

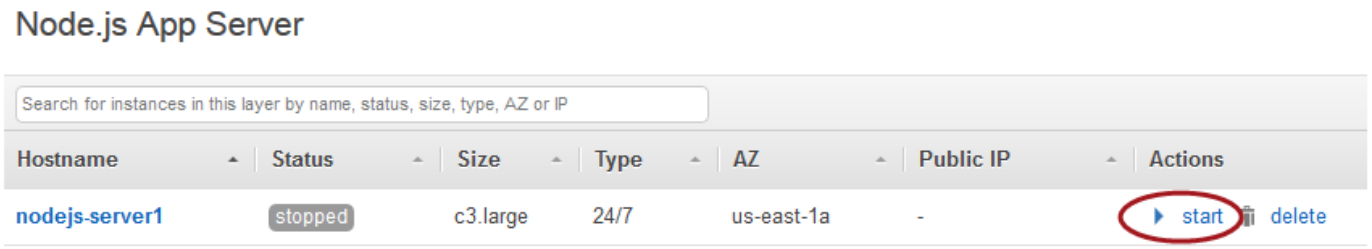
   - In the service navigation pane, choose **Instances**:



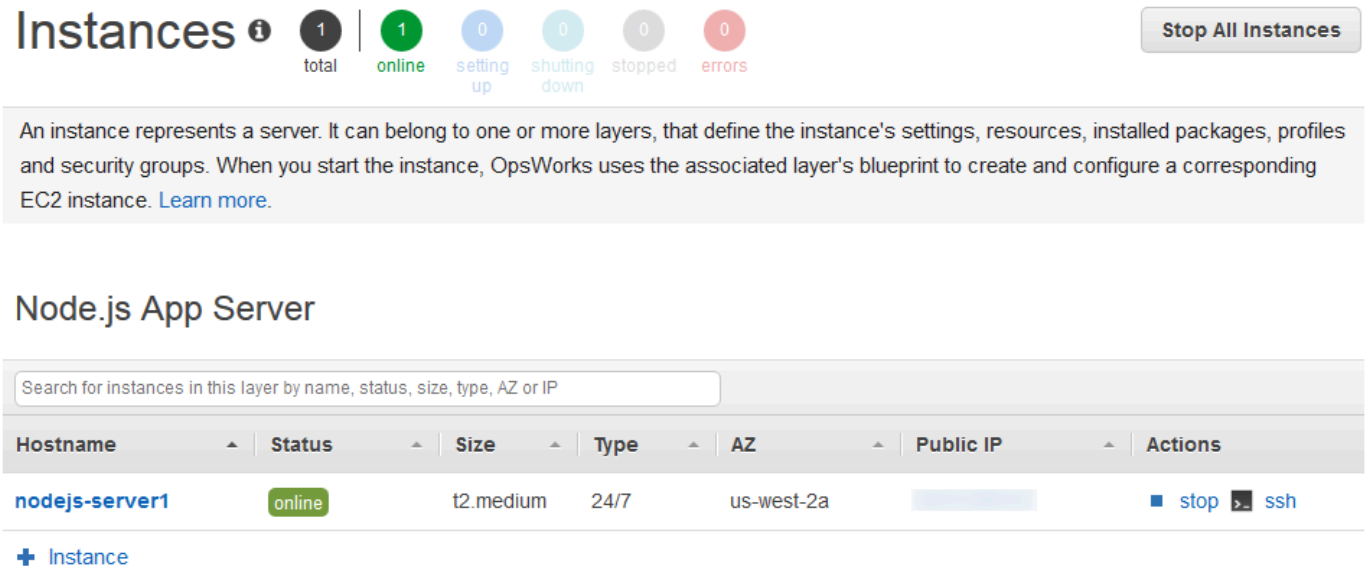
   - On the **My Sample Stack (Linux)** page, choose **Instances**:



2. On the **Instances** page, for **Node.js App Server**, for **nodejs-server1**, choose **start**:



3. Do not proceed until the **online** circle is bright green. (If you see a failure message, consult the Debugging and Troubleshooting Guide.)
4. As the instance is setting up, OpsWorks Stacks deploys the app to the instance.
5. Your results should resemble the following screenshot before you continue (if you receive a failure message, you may want to consult the Debugging and Troubleshooting Guide.):



You now have an instance with an app that has been deployed to the instance.

In the next step, you test the app on the instance.

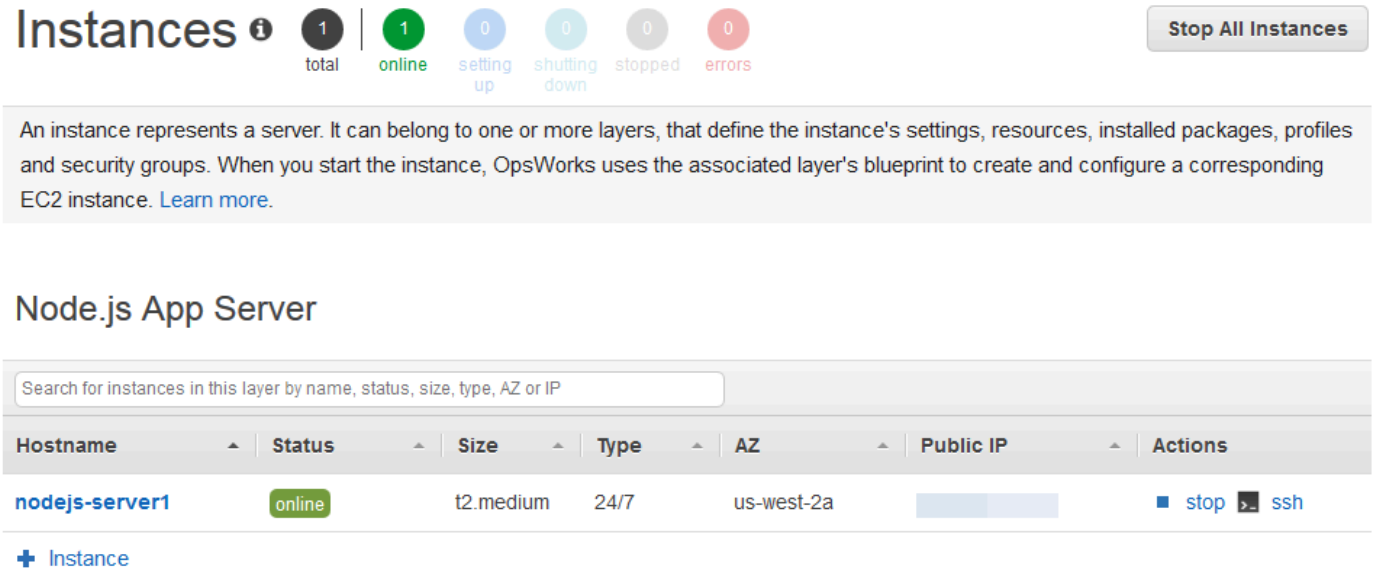
Step 4: Test the Deployed App on the Instance

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Test the results of the app deployment on the instance.

To test the deployment on the instance

1. With the **Instances** page displayed from the previous step, for **Node.js App Server**, for **nodejs-server1**, for **Public IP**, choose the IP address.



2. On the congratulatory web page, in the **Leave a comment** text box, type a comment, and then choose **Send** to test the app. The app adds your comment to the web page. Continue leaving comments and choosing **Send** as often as you want.



3. If you have a Twitter account, choose **Tweet** or **Follow @AWSOpsWorks**, and follow the on-screen directions to tweet about the app or to follow @AWSOpsWorks.

You have now successfully tested the deployed app on the instance.

In the remaining steps, you can use the OpsWorks Stacks console to explore settings of the stack and its components. In the next step, you can start your exploration by examining the stack's settings.

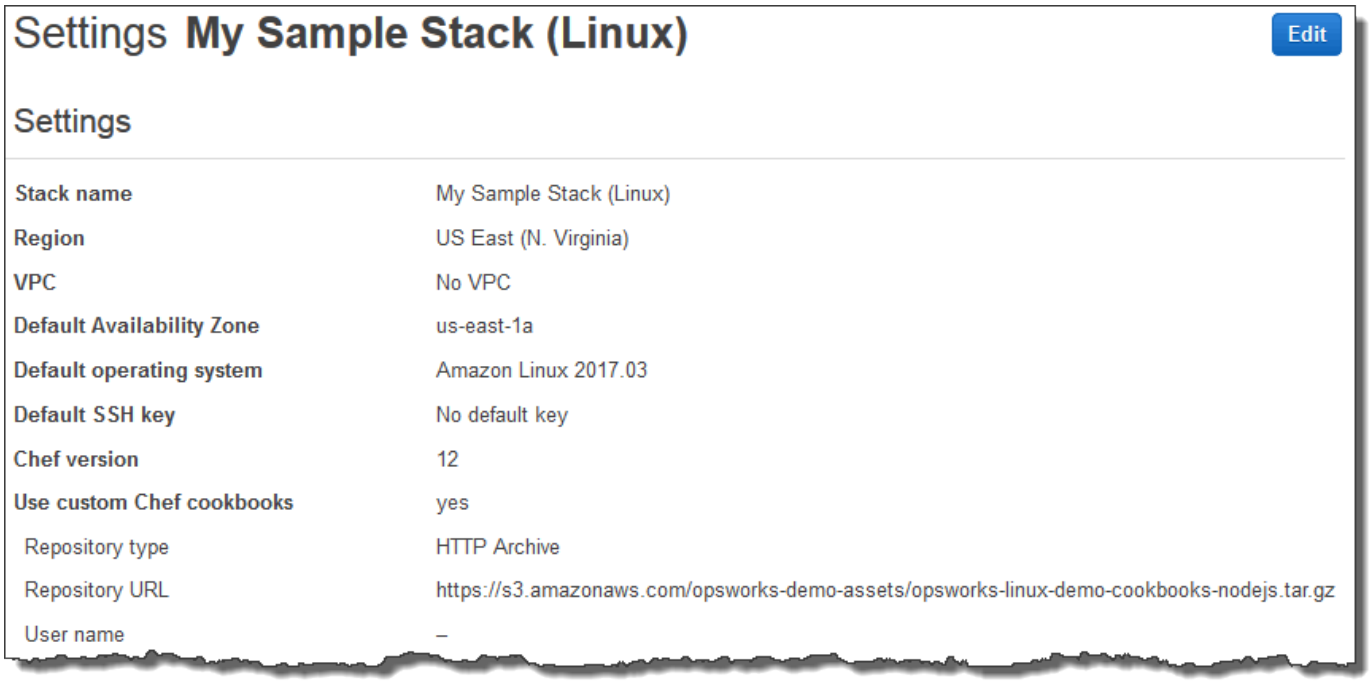
Step 5: Explore the Stack's Settings

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Examine how OpsWorks Stacks set up the stack.

To display the stack's settings

1. In the service navigation bar, choose **Stack**. The **My Sample Stack (Linux)** page displays.
2. Choose **Stack Settings**. The **Settings My Sample Stack (Linux)** page displays:



To learn more about many of the settings, choose **Edit**, and then hover over each of the settings. (Not all settings have on-screen descriptions.) For more information about these settings, see Create a New Stack.

To explore the Chef cookbook used in this walkthrough, open the opsworks-linux-demo-cookbooks-nodejs repository on GitHub.

In the next step, you can explore the layer's settings.

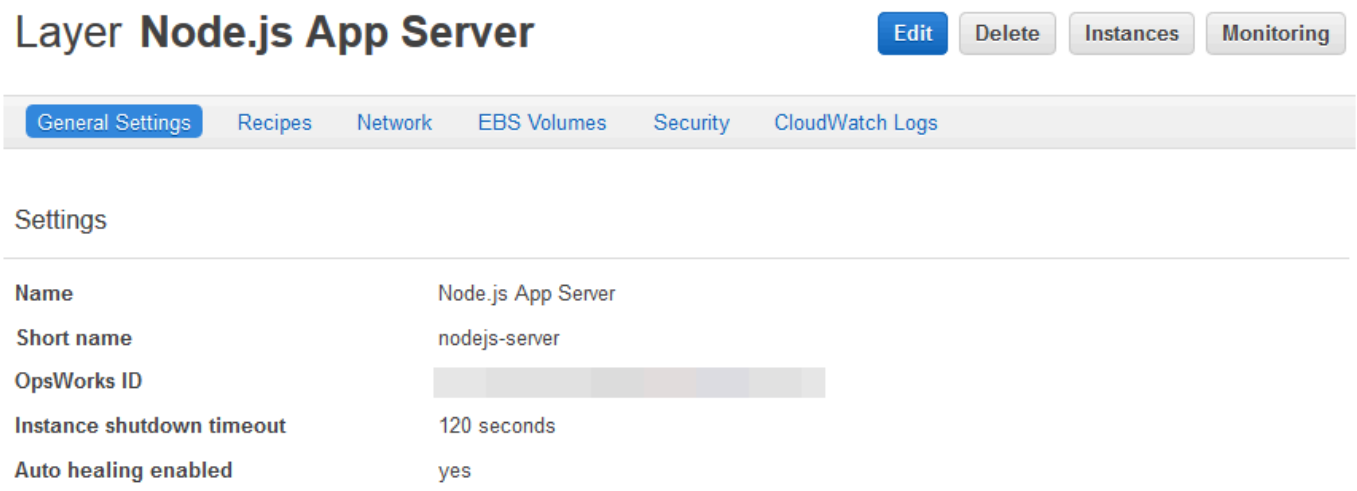
Step 6: Explore the Layer's Settings

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Examine how OpsWorks Stacks set up the layer.

To display the layer's settings

1. In the service navigation pane, choose **Layers**. The **Layers** page is displayed.
2. Choose **Node.js App Server**. The **Layer Node.js App Server** page is displayed. To view the layer's settings, choose **General Settings**, **Recipes**, **Network**, **EBS Volumes**, and **Security**:



To learn more about many of the settings, choose **Edit**, and then hover over each of the settings. (Not all settings have on-screen descriptions.) For more information about these settings, see Editing an OpsWorks Layer's Configuration.

In the next step, you can explore the instance's settings and logs.

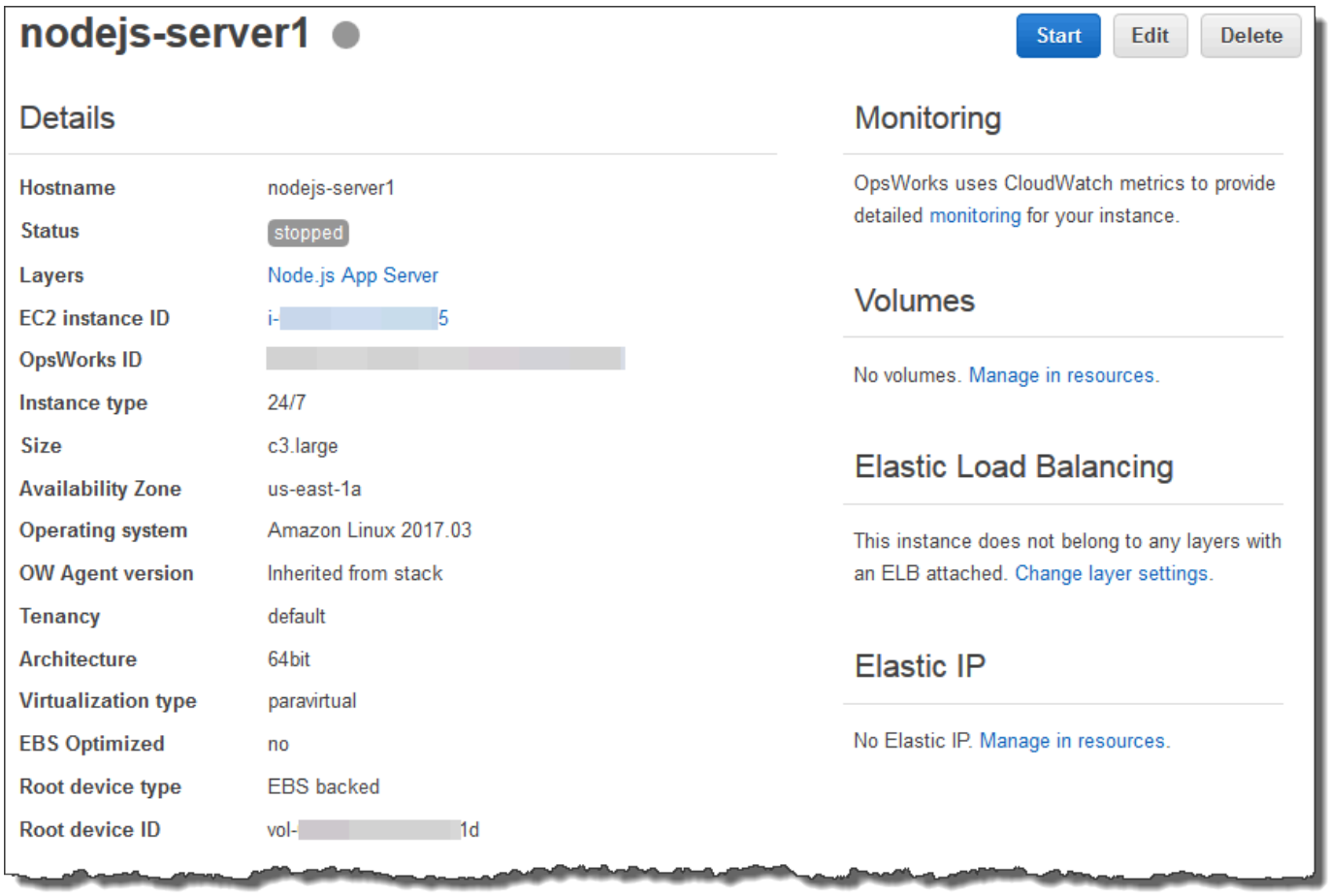
Step 7: Explore the Instance's Settings and Logs

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

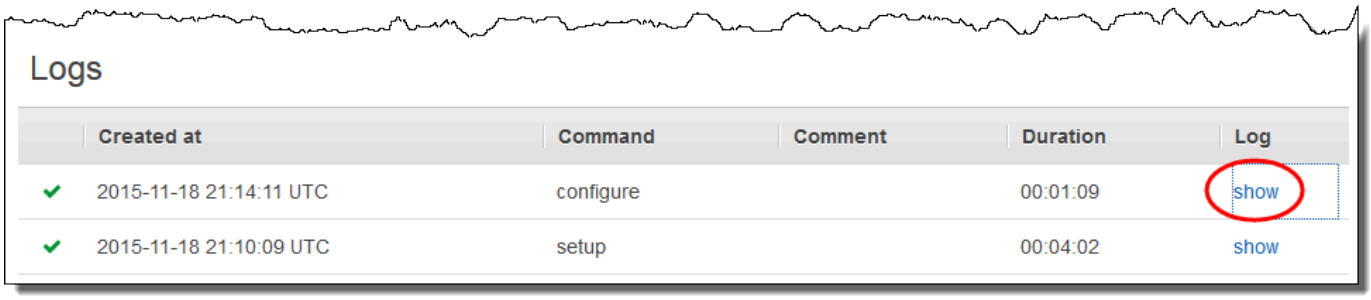
Examine the settings that OpsWorks Stacks used to launch the instance. You can also examine the instance logs that OpsWorks Stacks created.

To display the instance's settings and logs

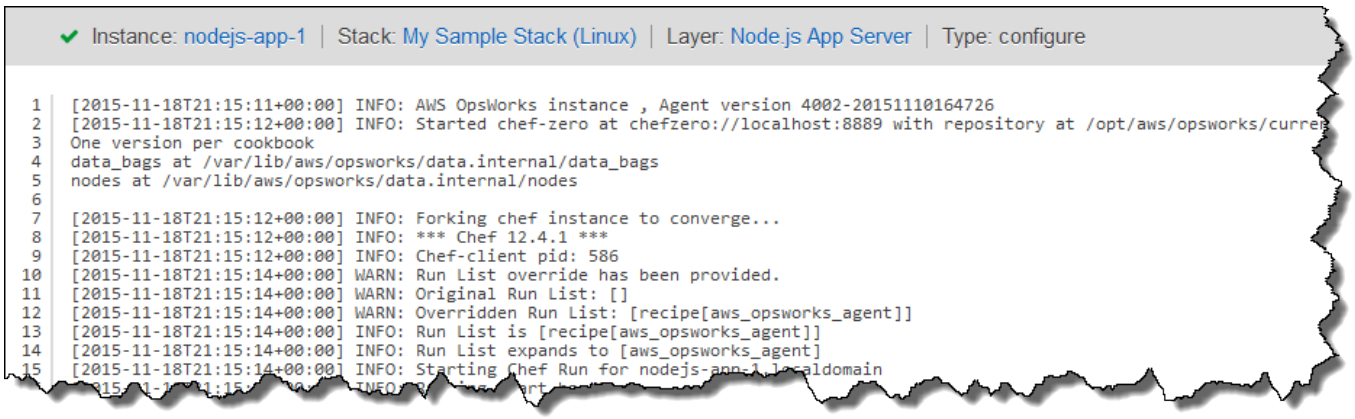
1. In the service navigation pane, choose **Instances**. The **Instances** page displays.
2. For **Node.js App Server**, choose **nodejs-server1**. The instance's properties page is shown.



3. To explore the instance logs, in the **Logs** section, for **Log**, choose **show**.



4. OpsWorks Stacks displays the log in a separate web browser tab.



To learn more about what some of the instance settings represent, return to the **nodejs-server1** page, choose **Stop**, and when you see the confirmation message, choose **Stop**. Choose **Edit** after **Status** changes from **stopping** to **stopped**, and then hover over each of the settings. (Not all settings have on-screen descriptions.) For more information about these settings, see Adding an Instance to a Layer.

When you have finished reviewing settings, choose **Start** to restart the instance, and wait until **Status** changes to **online**. Otherwise, you won't be able to test the app later, because the instance will remain stopped.

> **Note**
>
> If you want to log in to the instance to explore it further, you must first provide OpsWorks Stacks with information about your public SSH key (which you can create with tools such as ssh-keygen or PuTTYgen), and then you must set permissions on the **My Sample Stack (Linux)** stack to enable your user to log in to the instance. For instructions, see Registering a User's Public SSH Key and Logging In with SSH.

In the next step, explore the app's settings.

Step 8: Explore the App's Settings

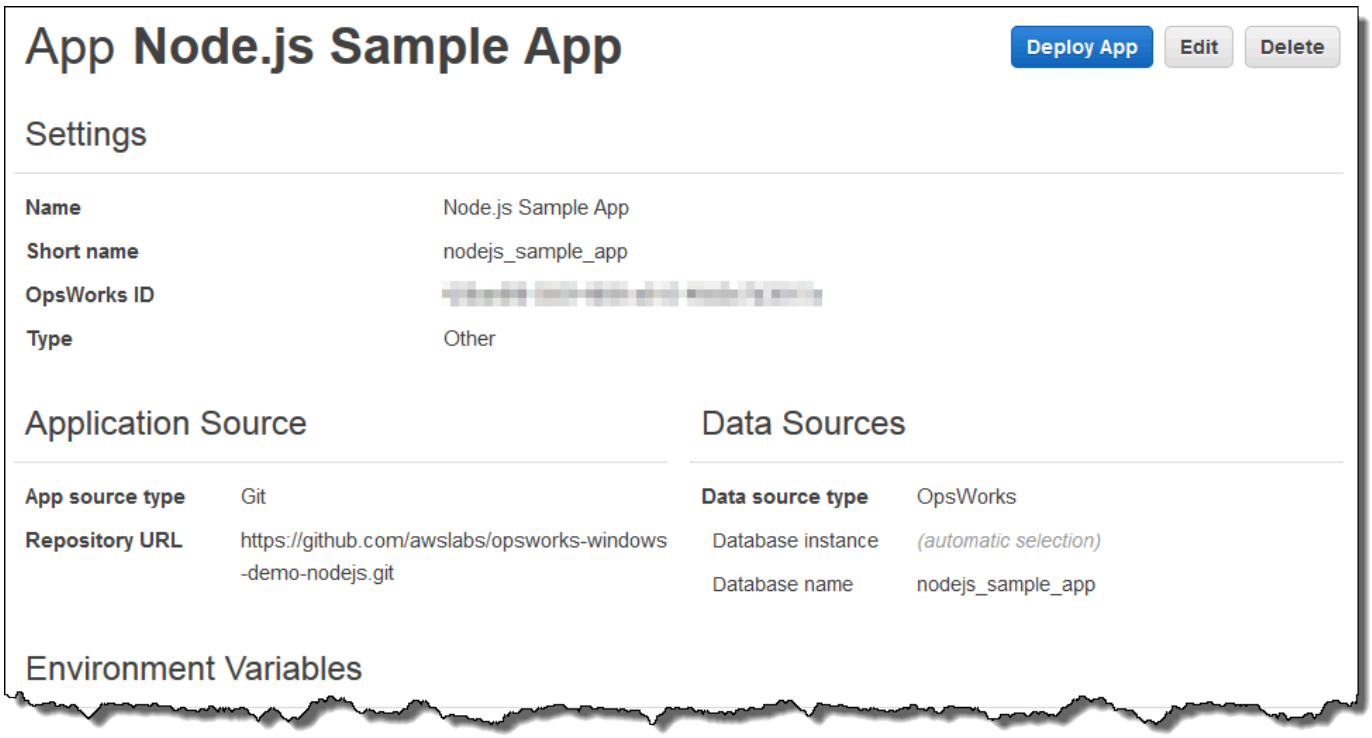
> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate

their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Examine the settings that OpsWorks Stacks used for the app.

To display the app's settings

1. In the service navigation pane, choose **Apps**. The **Apps** page is displayed.
2. Choose **Node.js Sample App**. The **App Node.js Sample App** page displays:



To learn about what some of the settings represent, choose **Edit**, and then hover over each of the settings. (Not all settings have on-screen descriptions.) For more information about these the settings, see Adding Apps.

In the next step, you can explore layer monitoring reports.

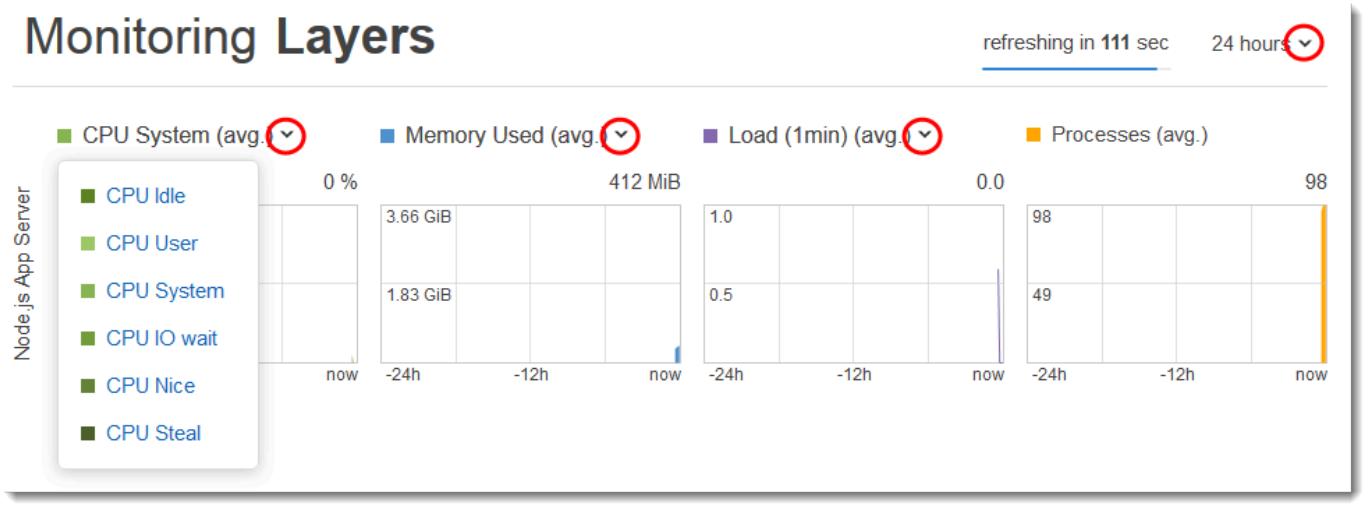
Step 9: Explore Layer Monitoring Reports

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Examine reports that OpsWorks Stacks generates about the layer's compute performance.

To display layer monitoring reports

1. In the service navigation pane, choose **Monitoring**. The **Monitoring Layers** page is displayed.
2. To explore additional views, choose the arrow next to **CPU**, **Memory**, **Load**, and time:



For information about these and other reports, see Using Amazon CloudWatch and Monitoring.

In the next step, you can explore additional stack settings.

Step 10: Explore Additional Stack Settings

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

In this step, you can examine additional stack settings.

OpsWorks Stacks ran no separate deployments, provisioned no additional resources, and adjusted no additional permissions as part of this stack, so there isn't much of interest on the **Deployments and Commands**, **Resources**, and **Permissions** pages. If you want to see those settings anyway, choose **Deployments**, **Resources**, and **Permissions** in the service navigation pane, respectively. If you want more information about what these pages represent, see Deploying Apps, Resource Management, and Managing User Permissions.

In the next step, you can clean up the AWS resources that you used for this walkthrough. This next step is optional. You may want to keep using these AWS resources as you continue to learn more about OpsWorks Stacks. However, keeping these AWS resources around may result in some ongoing charges to your AWS account. If you want to keep these AWS resources around for later use, you have now completed this walkthrough, and you can skip ahead to Next Steps.

Step 11 (Optional): Clean Up

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

To prevent incurring additional charges to your AWS account, you can delete the app and the AWS resources that were used for this walkthrough, including the instance and the OpsWorks Stacks

stack. (For more information, see OpsWorks Pricing.) However, you might want to keep using these AWS resources as you continue to learn more about OpsWorks Stacks. If you want to keep these AWS resources available, you have now completed this walkthrough, and you can skip to Next Steps.

Content stored in the resources that you created for this walkthrough can contain personally-identifying information. If you no longer want this information to be stored by AWS, follow steps in this topic.

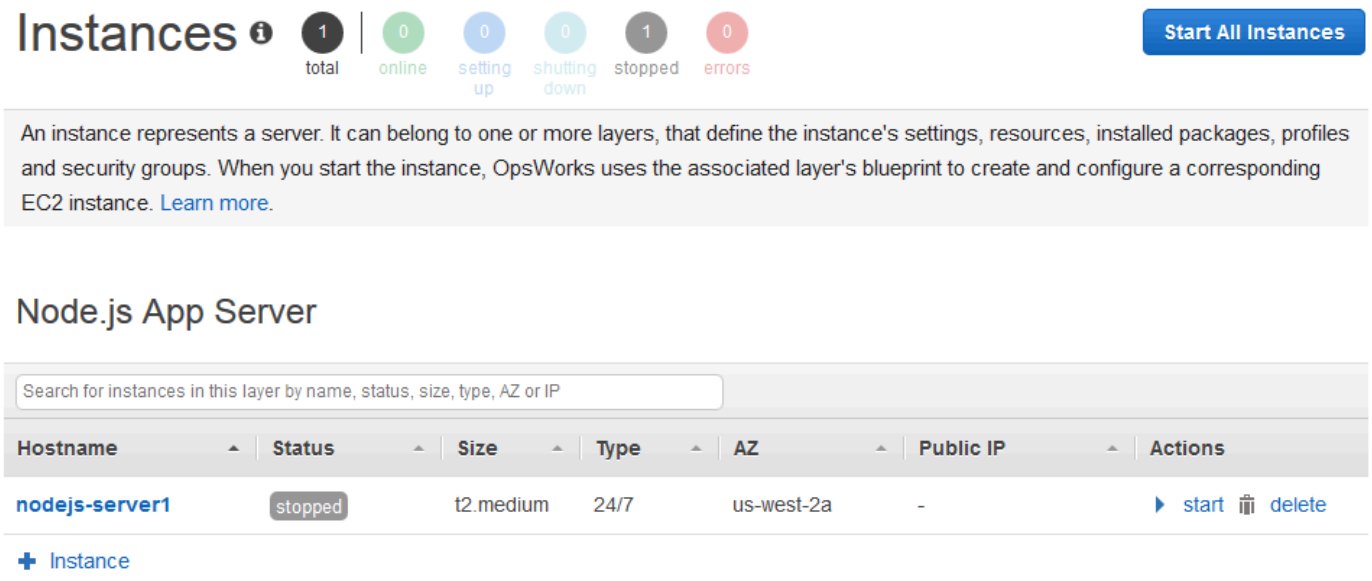
To delete the app from the stack

1. In the service navigation pane, choose **Apps**. The **Apps** page displays.
2. For **Node.js Sample App**, for **Actions**, choose **delete**. When you see the confirmation message, choose **Delete**. When the app is deleted, you see the **No apps** message.

To delete the instance for the stack

1. In the service navigation pane, choose **Instances**. The **Instances** page displays.
2. For **Node.js App Server**, for **nodejs-server1**, for **Actions**, choose **stop**. When you see the confirmation message, choose **Stop**.

   This process can take a few minutes. When OpsWorks Stacks is finished, the following results are shown.



3. For **Actions**, choose **delete**. When you see the confirmation message, choose **Delete**. The instance is deleted, and the **No instances** message is displayed.

To delete the stack

1. In the service navigation pane, choose **Stack**. The **My Sample Stack (Linux)** page is displayed.
2. Choose **Delete Stack**. When you see the confirmation message, choose **Delete**. The stack is deleted, and the **OpsWorks Dashboard** page is displayed.

Optionally, you can delete the user and Amazon EC2 key pair that you used for this walkthrough, if you don't want to reuse them for access to other AWS services and EC2 instances. For instructions, see Deleting an IAM user and Amazon EC2 key pairs and Linux instances.

You have now completed this walkthrough. For more information, see Next Steps.

Next Steps

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Now that you have completed this walkthrough, you can learn more about using OpsWorks Stacks:

- Practice manually recreating this stack by yourself using OpsWorks Stacks. See Getting Started: Linux.
- Explore the cookbook and the app that OpsWorks Stacks used for this walkthrough. See Learning More: Explore the Cookbook Used in This Walkthrough and Learning More: Explore the App Used in This Walkthrough in the companion Getting Started: Linux walkthrough.
- Practice using OpsWorks Stacks with Windows instances. See Getting Started: Windows.
- Learn more about stacks by learning how to Create a New Stack.
- Learn more about layers by Editing an OpsWorks Layer's Configuration.

- Learn more about instances by Adding an Instance to a Layer.
- Learn more about apps by Deploying Apps.
- Learn more about Cookbooks and Recipes.
- Create your own cookbooks. See Getting Started: Cookbooks.
- Learn about controlling access to stacks with Security and Permissions.

Getting Started with Linux Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

In this walkthrough, you will learn how to use OpsWorks Stacks to create a Node.js application environment. When you are done, you will have an Amazon Elastic Compute Cloud (Amazon EC2) instance running Chef 12, a Node.js HTTP server, and a web app that you can use to interact with Twitter and leave comments on a web page.

Chef is a third-party framework for configuring and maintaining servers, such as EC2 instances, and how apps are deployed and maintained on those servers. If you aren't familiar with Chef, after completing this walkthrough, we recommend that you learn more about Chef so that you can take full advantage of all that OpsWorks Stacks has to offer. (For more information, see the Learn Chef website.)

OpsWorks Stacks supports four Linux distributions: Amazon Linux, Ubuntu Server, CentOS, and Red Hat Enterprise Linux. For this walkthrough, we use Ubuntu Server. OpsWorks Stacks also works with Windows Server. Although we have an equivalent walkthrough for Windows Server stacks, we recommend that you complete this walkthrough first to learn basic concepts about OpsWorks Stacks and Chef that are not repeated there. After you complete this walkthrough, see the Getting Started: Windows walkthrough.

Topics

- Step 1: Complete the Prerequisites
- Step 2: Create a Stack
- Step 3: Add a Layer to the Stack
- Step 4: Specify the App to Deploy to the Instance
- Step 5: Launch an Instance
- Step 6: Deploy the App to the Instance
- Step 7: Test the Deployed App on the Instance
- Step 8 (Optional): Clean Up
- Next Steps
- Learning More: Explore the Cookbook Used in This Walkthrough
- Learning More: Explore the App Used in This Walkthrough

Step 1: Complete the Prerequisites

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Complete the following setup steps before you can start the walkthrough. These setup steps include signing up for an AWS account, creating an administrative user, and assigning access permissions to OpsWorks Stacks.

If you have already completed the Getting Started: Sample walkthrough, then you have met the prerequisites for this walkthrough, and you can skip ahead to Step 2: Create a Stack.

Topics

- Sign up for an AWS account
- Create a user with administrative access
- Assign Service Access Permissions

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.
2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.
2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Assign Service Access Permissions

Enable access to the OpsWorks Stacks service (and related services that OpsWorks Stacks relies on) by adding the `AWSOpsWorks_FullAccess` and `AmazonS3FullAccess` permissions to your role or user.

For more information about adding permissions, see Adding IAM identity permissions (console).

You have now completed all of the setup steps and can start this walkthrough.

Step 2: Create a Stack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You will use the OpsWorks Stacks console to create a stack. A *stack* is a collection of instances and related AWS resources that have a common purpose and that you want to manage together. (For more information, see Stacks.) For this walkthrough, there is only one instance.

Before you begin, complete the prerequisites if you haven't already.

To create the stack

1. Sign in to the AWS Management Console and open the OpsWorks console at https://console.aws.amazon.com/opsworks/.
2. Do any of the following, if they apply:
   - If the **Welcome to OpsWorks Stacks** page is displayed, choose **Add your first stack** or **Add your first OpsWorks Stacks stack** (both choices do the same thing). The **Add stack** page is displayed.
   - If the **OpsWorks Dashboard** page is displayed, choose **Add stack**. The **Add stack** page is displayed.
3. With the **Add stack** page displayed, choose **Chef 12 stack** if it is not already chosen for you.
4. In the **Stack name** box, type a name, for example **`MyLinuxDemoStack`**. (You can type a different name, but be sure to substitute it for `MyLinuxDemoStack` throughout this walkthrough.)
5. For **Region**, choose **US West (Oregon)**.
6. For **VPC**, do one of the following:
   - If a VPC is available, choose it. (For more information, see Running a Stack in a VPC.)
   - Otherwise, choose **No VPC**.

7. For **Default operating system**, choose **Linux** and **Ubuntu 18.04 LTS**.
8. For **Use custom Chef cookbooks**, choose **Yes**.
9. For **Repository type**, choose **Http Archive**.
10. For **Repository URL**, type **`https://s3.amazonaws.com/opsworks-demo-assets/opsworks-linux-demo-cookbooks-nodejs.tar.gz`**
11. Leave the defaults for the following:

    - **Default Availability Zone** (**us-west-2a**)
    - **Default SSH key** (**Do not use a default SSH key**)
    - **User name** (blank)
    - **Password** (blank)
    - **Stack color** (dark blue)
12. Choose **Advanced**.
13. For **IAM role**, do one of the following (for more information, see Allowing OpsWorks Stacks to Act on Your Behalf):

    - If **aws-opsworks-service-role** is available, choose it.
    - If **aws-opsworks-service-role** is not available, choose **New IAM role**.
14. For **Default IAM instance profile**, do one of the following (for more information, see Specifying Permissions for Apps Running on EC2 instances):

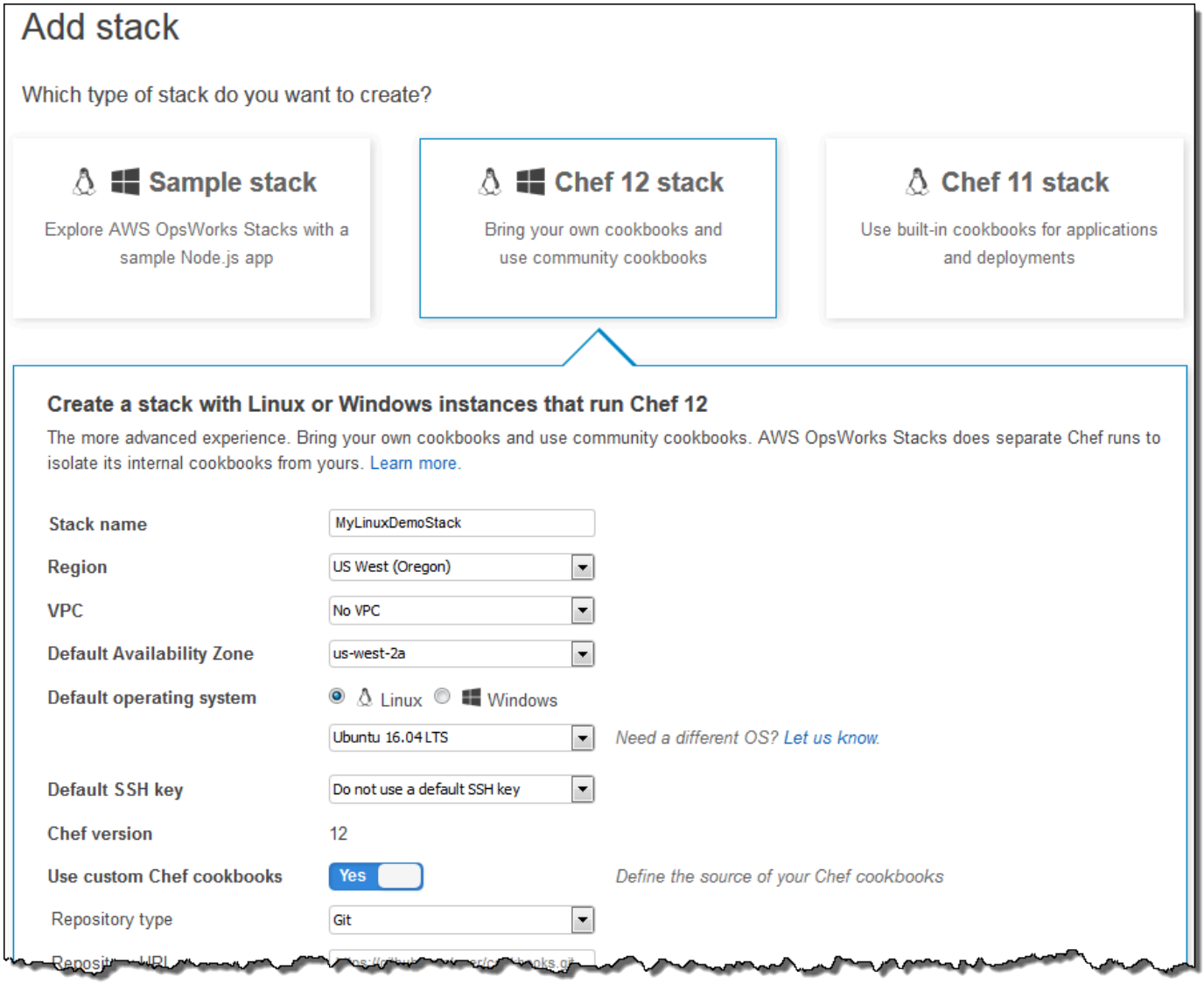
    - If **aws-opsworks-ec2-role** is available, choose it.
    - If **aws-opsworks-ec2-role** is not available, choose **New IAM instance profile**.
15. For **API endpoint region**, choose the regional API endpoint with which you want the stack to be associated. If you want the stack to be in the US West (Oregon) Region within the US East (N. Virginia) regional endpoint, choose **us-east-1**. If you want the stack to be both in the US West (Oregon) Region and associated with the US West (Oregon) regional endpoint, choose **us-west-2**.

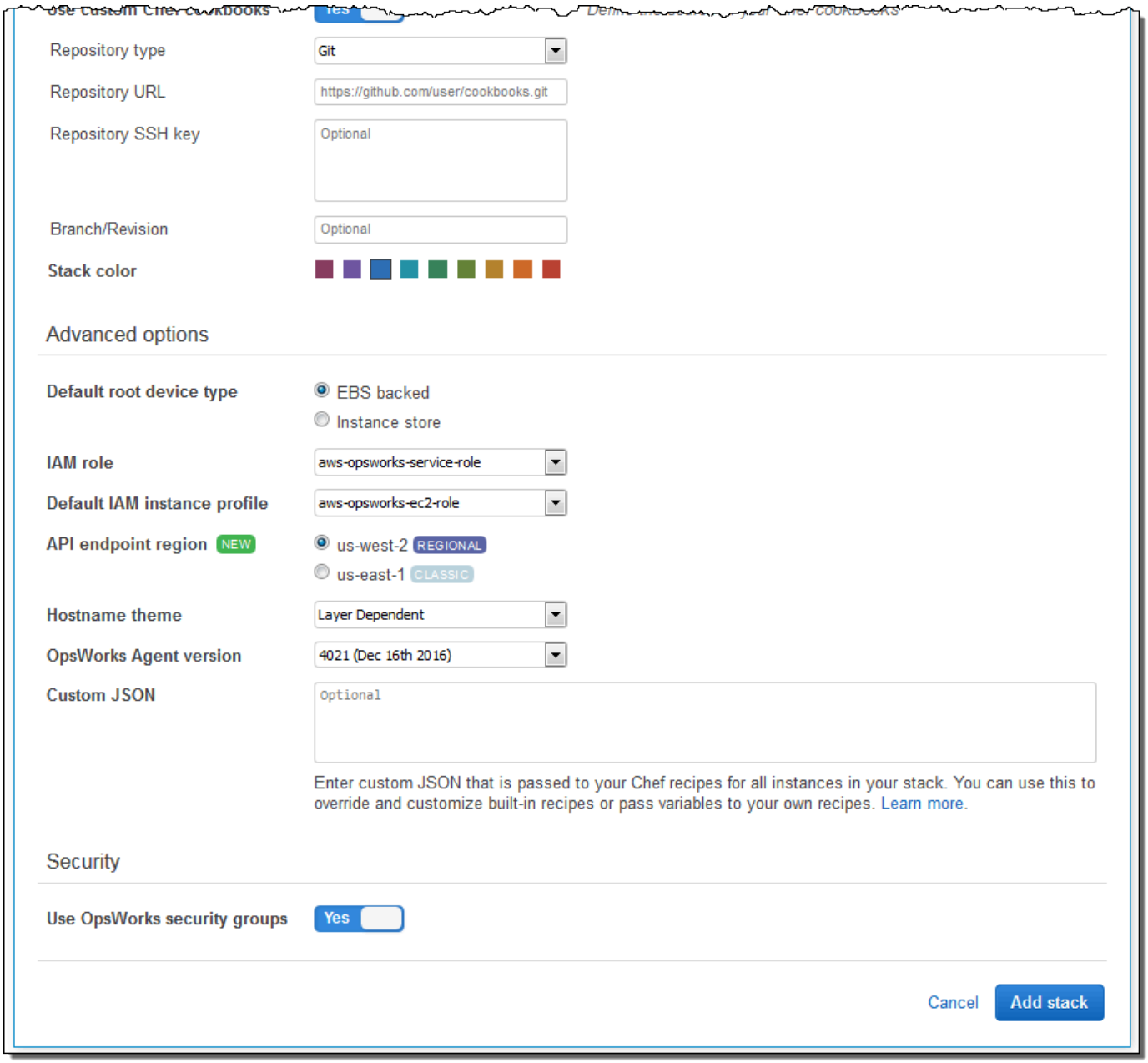
    > **Note**
    >
    > The US East (N. Virginia) regional endpoint includes older AWS Regions for backward compatibility, but it is a best practice to choose the regional endpoint that is closest to where you manage AWS. For more information, see Region Support.

16. Leave the defaults for the following:

    - **Default root device type** (**EBS backed**)
    - **Hostname theme** (**Layer Dependent**)
    - **OpsWorks Agent version** (most recent version)
    - **Custom JSON** (blank)
    - **Use OpsWorks security groups** (**Yes**)

17. Your results should match the following screenshots except for perhaps **VPC**, **IAM role**, and **Default IAM instance profile**:





18. Choose **Add Stack**. OpsWorks Stacks creates the stack and displays the **MyLinuxDemoStack** page.

You now have a stack with the correct settings for this walkthrough.

In the next step, you will add a layer to the stack.

Step 3: Add a Layer to the Stack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

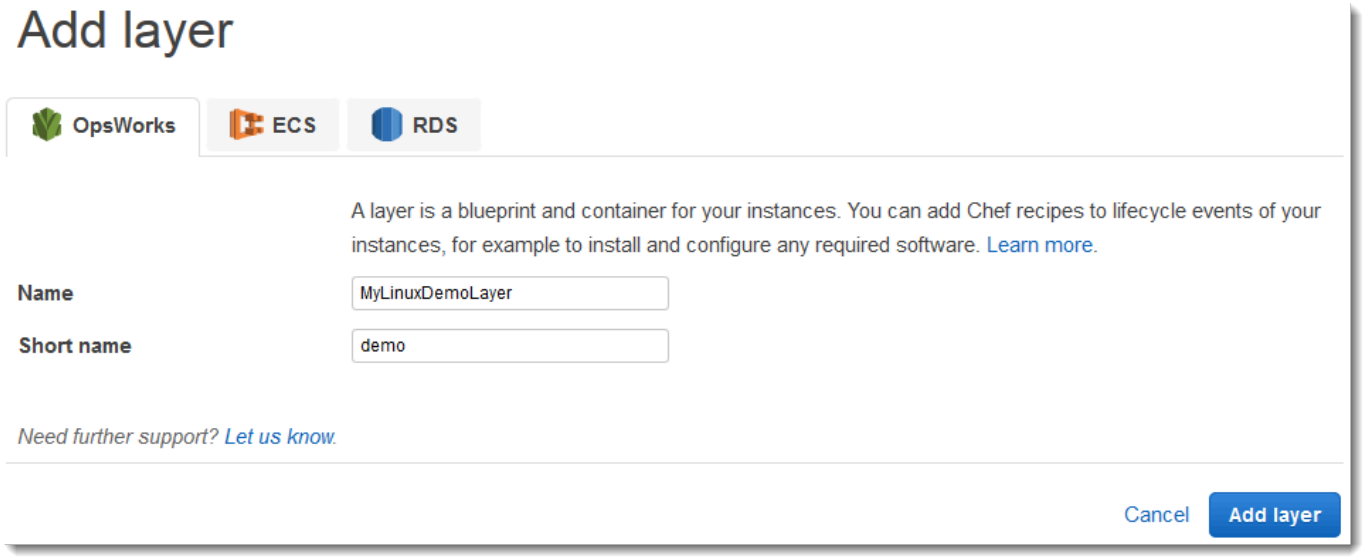
A *layer* is a blueprint for a set of instances, such as Amazon EC2 instances. It specifies information such as the instance's settings, resources, installed packages, and security groups. Next, add a layer to the stack. (For more information about layers, see Layers.)

To add the layer to the stack

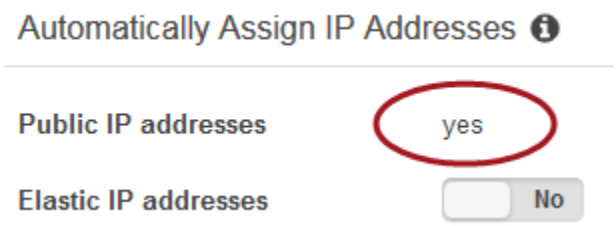
1. With the **MyLinuxDemoStack** page displayed from the previous step, for **Layers**, choose **Add a layer**:



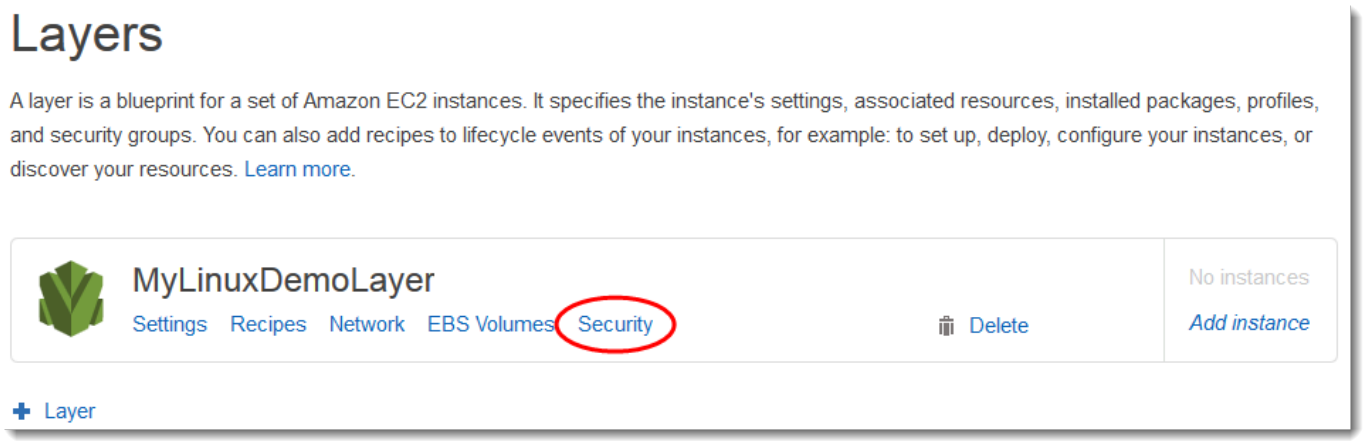
2. The **Add Layer** page is displayed. On the **OpsWorks** tab, for **Name**, type **`MyLinuxDemoLayer`**. (You can type a different name, but be sure to substitute it for `MyLinuxDemoLayer` throughout this walkthrough.)
3. For **Short name**, type **`demo`** (you can type a different value, but be sure to substitute it for `demo` throughout this walkthrough):



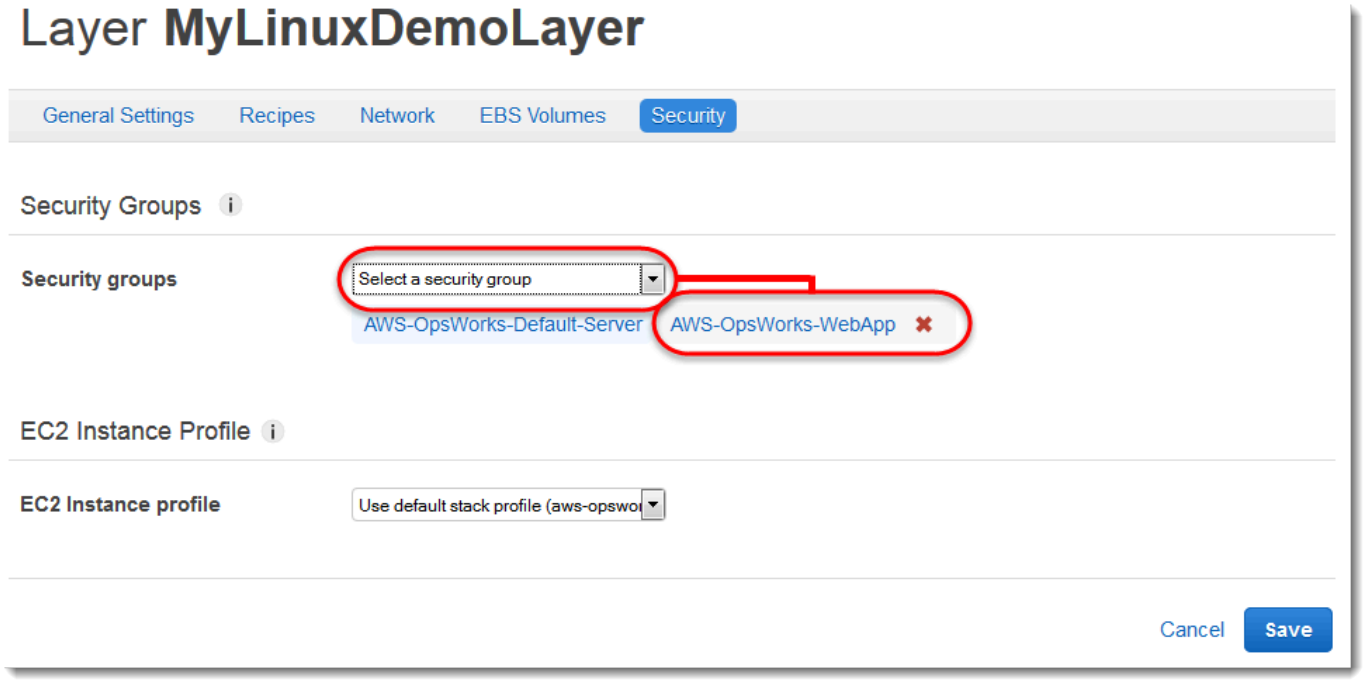
4. Choose **Add layer**. OpsWorks Stacks creates the layer and displays the **Layers** page.
5. On the **Layers** page, for **MyLinuxDemoLayer**, choose **Network**.
6. On the **Network** tab, under **Automatically Assign IP Addresses**, verify that **Public IP addresses** is set to **yes**. If you've made changes, choose **Save**.



7. On the **Layers** page, choose **Security**:



8. The **Layer MyLinuxDemoLayer** page is displayed with the **Security** tab open. For **Security groups**, choose **AWS-OpsWorks-WebApp**, and then choose **Save**:



9. The `AWS-OpsWorks-WebApp` security group is added to the layer. (This security group enables users to connect to the app on the instance later in this walkthrough. Without this security group, users will receive a message in their web browser that they cannot connect to the instance.)

You now have a layer with the correct settings for this walkthrough.

In the next step, you will specify the app to deploy to the instance.

Step 4: Specify the App to Deploy to the Instance

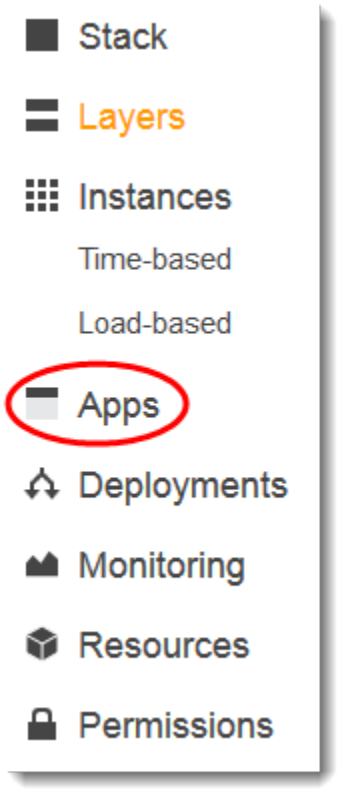
> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Tell OpsWorks Stacks about the app that you will deploy to the instance later in this walkthrough. In this context, OpsWorks Stacks defines an *app* as code you want to run on an instance. (For more information, see Apps.)

The procedure in this section applies to Chef 12 and newer stacks. For information about how to add apps to layers in Chef 11 stacks, see Step 2.4: Create and Deploy an App - Chef 11.

To specify the app to deploy

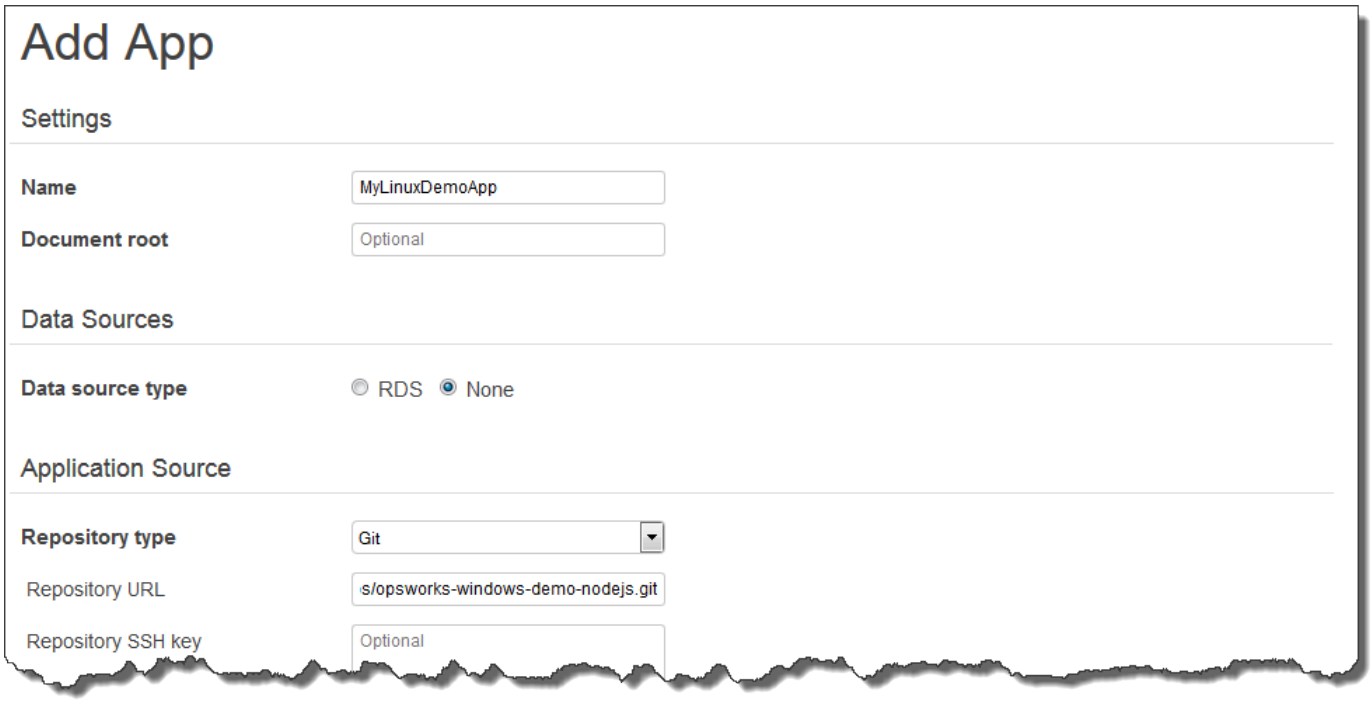
1. In the service navigation pane, choose **Apps**:

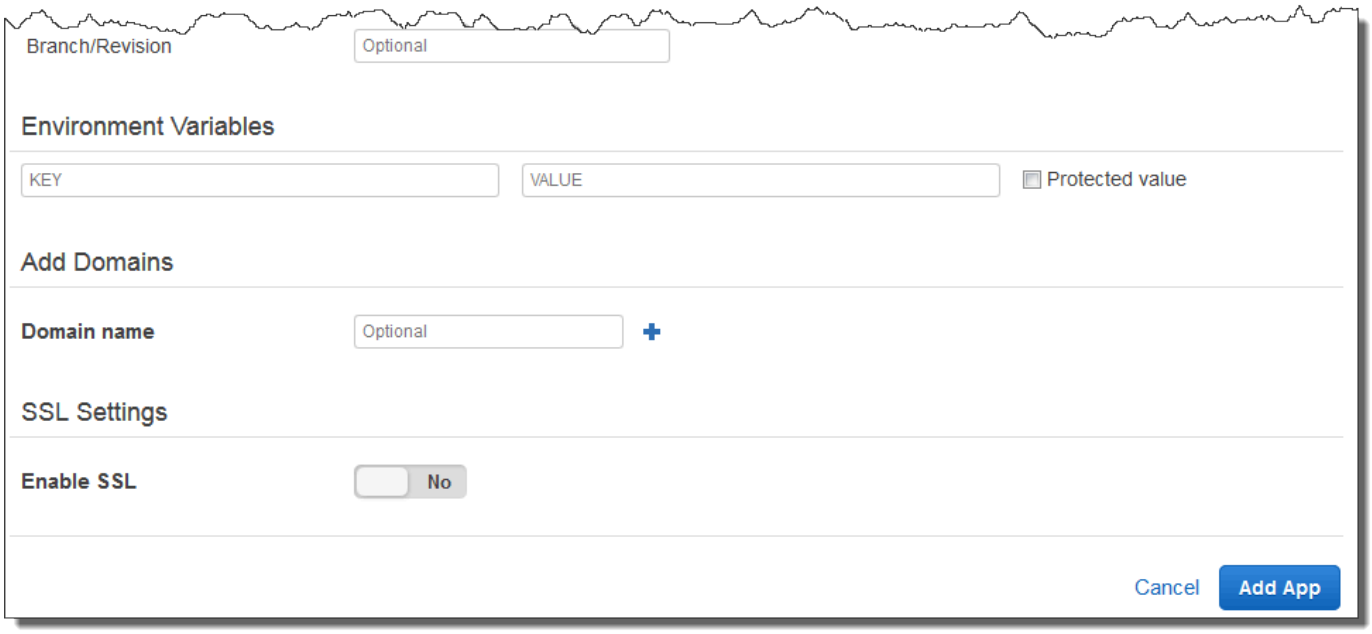


2. The **Apps** page is displayed. Choose **Add an app**. The **Add App** page is displayed.
3. For **Settings**, for **Name**, type **`MyLinuxDemoApp`**. (You can type a different name, but be sure to substitute it for `MyLinuxDemoApp` throughout this walkthrough.)
4. For **Application Source**, for **Repository URL**, type **`https://github.com/awslabs/opsworks-windows-demo-nodejs.git`**
5. Leave the defaults for the following:

   - **Settings**, **Document root** (blank)

- **Data Sources**, **Data source type** (**None**)
- **Repository type** (**Git**)
- **Repository SSH key** (blank)
- **Branch/Revision** (blank)
- **Environment Variables** (blank **KEY**, blank **VALUE**, unchecked **Protected Value**)
- **Add Domains**, **Domain Name** (blank)
- **SSL Settings**, **Enable SSL** (**No**)





6. Choose **Add App**. OpsWorks Stacks adds the app and displays the **Apps** page.

You now have an app with the correct settings for this walkthrough.

In the next step, you will launch the instance.

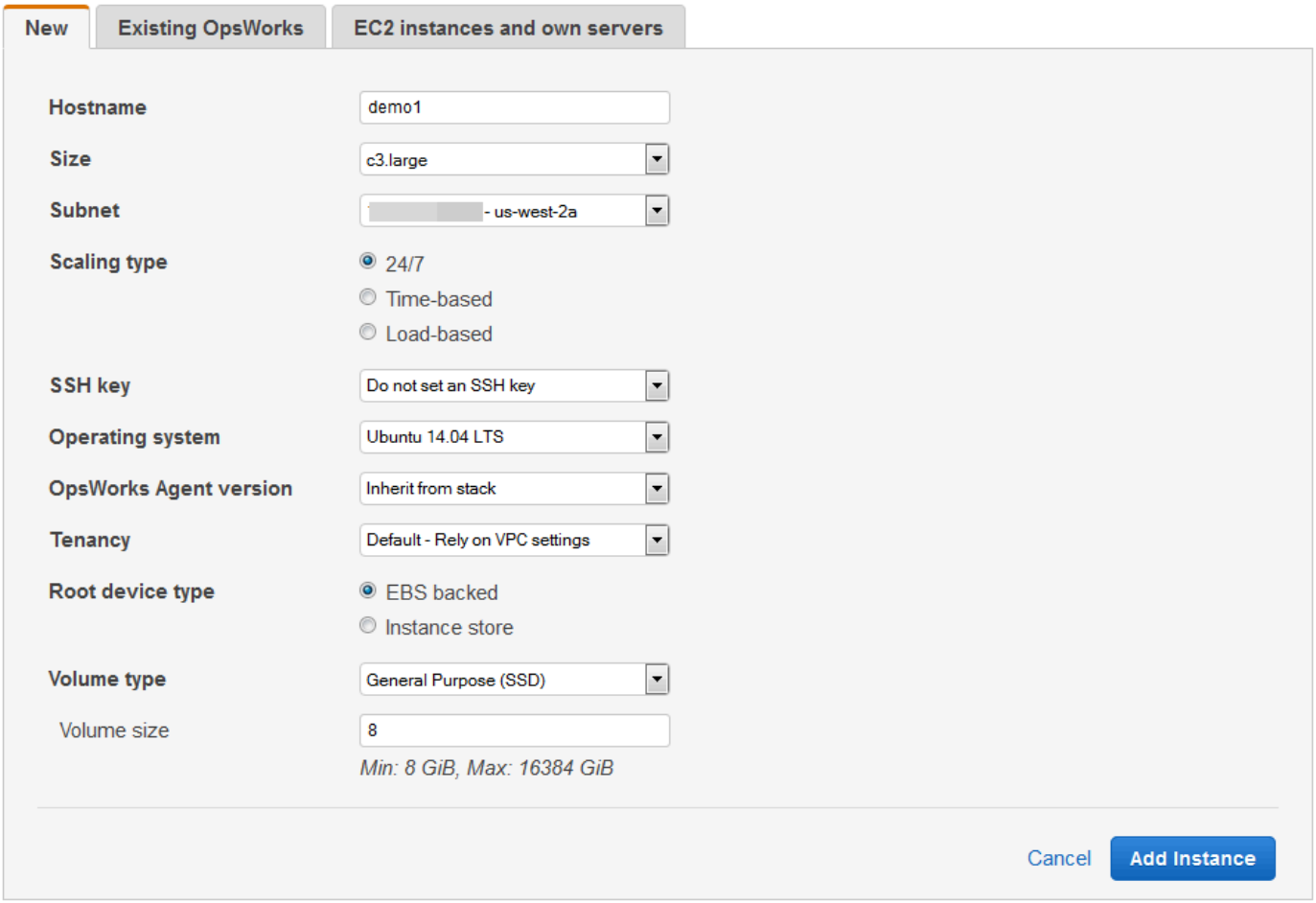
Step 5: Launch an Instance

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

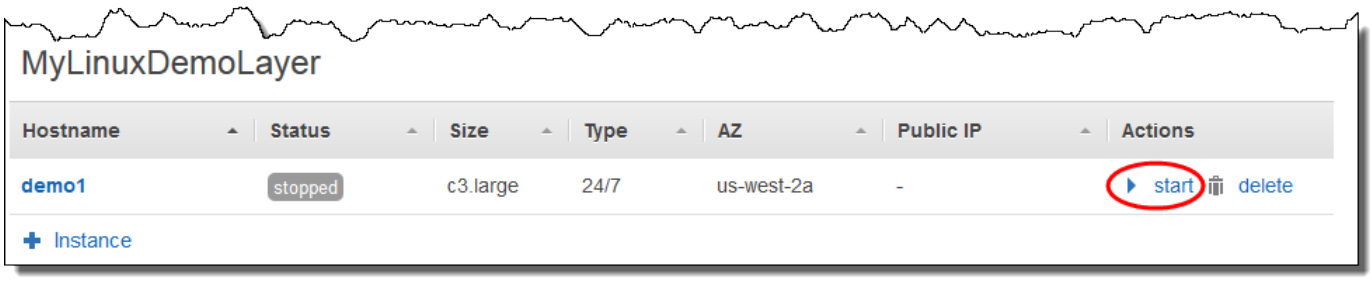
Use OpsWorks Stacks to start up an Ubuntu Server Amazon EC2 instance. This instance uses the settings that you defined in the layer you created earlier in this walkthrough. (For more information, see Instances.)

To launch the instance

1. In the service navigation pane, choose **Instances**. The **Instances** page is displayed.
2. For **MyLinuxDemoLayer**, choose **Add an instance**.
3. On the **New** tab, leave the defaults for the following:
   - **Hostname** (**demo1**)
   - **Size** (**c3.large**)
   - **Subnet** (*IP address* **us-west-2a**)
4. Choose **Advanced**.
5. Leave the defaults for the following:
   - **Scaling type** (**24/7**)
   - **SSH key** (**Do not use a default SSH key**)
   - **Operating system** (**Ubuntu 18.04 LTS**)
   - **OpsWorks Agent version** (**Inherit from stack**)
   - **Tenancy** (**Default - Rely on VPC settings**)
   - **Root device type** (**EBS backed**)
   - **Volume type** (**General Purpose (SSD)**)
   - **Volume size** (**8**)
6. Your results will be similar to the following screenshot:



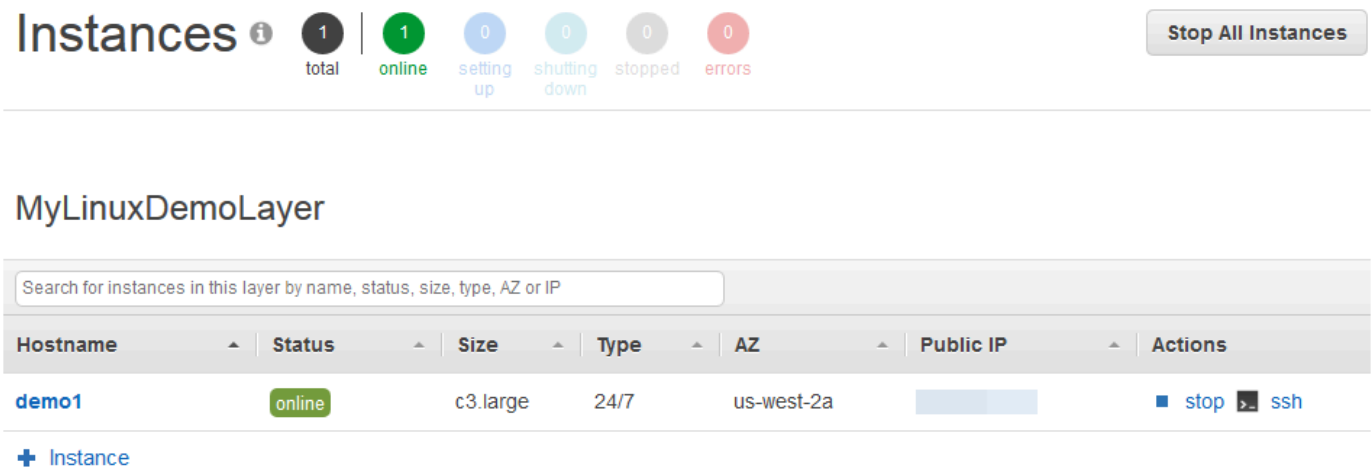
7. Choose **Add Instance**. OpsWorks Stacks adds the instance to the layer and displays the **Instances** page.
8. For **MyLinuxDemoLayer**, for **demo1**, for **Actions**, choose **start**:



9. Over the course of several minutes, the following occurs:
   - The **setting up** circle changes from **0** to **1**.

- **Status** turns from **stopped** to **requested**, to **pending**, to **booting**, to **running_setup**, and then finally to **online**. Note that this process can take several minutes.
- After **Status** changes to **online**, the **setting up** circle indicator changes from **1** to **0**, and the **online** circle changes from **0** to **1** and changes to bright green. Do not proceed until the **online** circle changes to bright green, and shows **1** instance online.

10. Your results must match the following screenshot before you continue (if you receive a failure message, you may want to consult the Debugging and Troubleshooting Guide):



You now have an instance that is ready for the app to be deployed to it.

> **Note**
>
> If you want to log in to the instance to explore it further, you must first provide OpsWorks Stacks with information about your public SSH key (which you can create with tools such as ssh-keygen or PuTTYgen), and then you must set permissions on the `MyLinuxDemoStack` stack to enable your user to log in to the instance. For instructions, see Registering a User's Public SSH Key and Logging In with SSH. If you plan to use SSH to connect to instances through PuTTY, see Connecting to Your Linux Instance from Windows Using PuTTY in the AWS documentation.

In the next step, you will deploy the app to the instance.

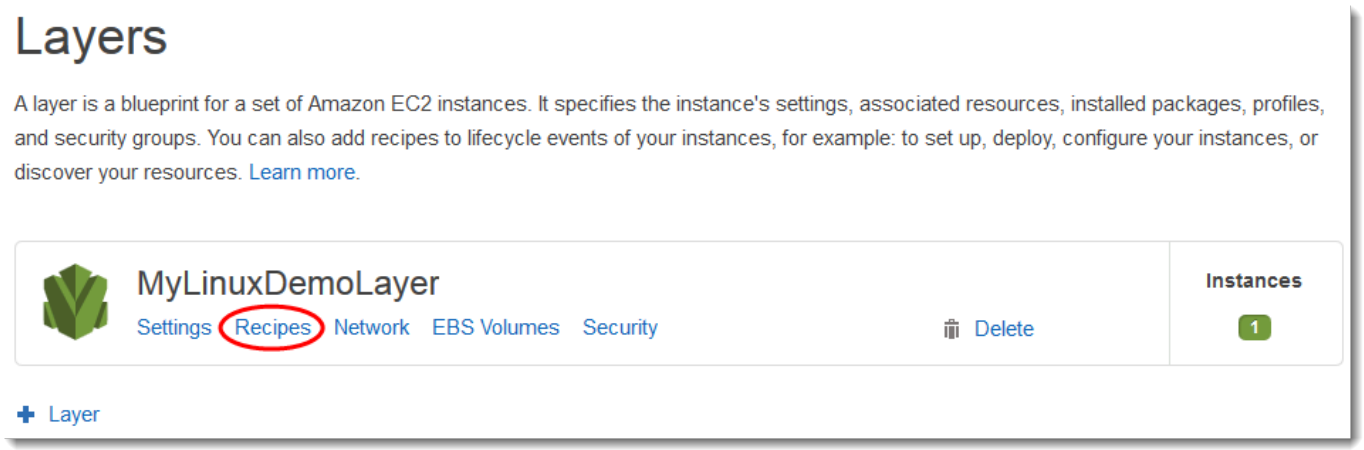
Step 6: Deploy the App to the Instance

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

In this step, you will deploy the app from GitHub to the running instance. (For more information, see Deploying Apps.) Before you deploy the app, you must specify the *recipe* to use to coordinate the deployment. A recipe is a Chef concept. Recipes are instructions, written with Ruby language syntax, that specify the resources to use and the order in which those resources are applied. (For more information, go to About Recipes on the Learn Chef website.)

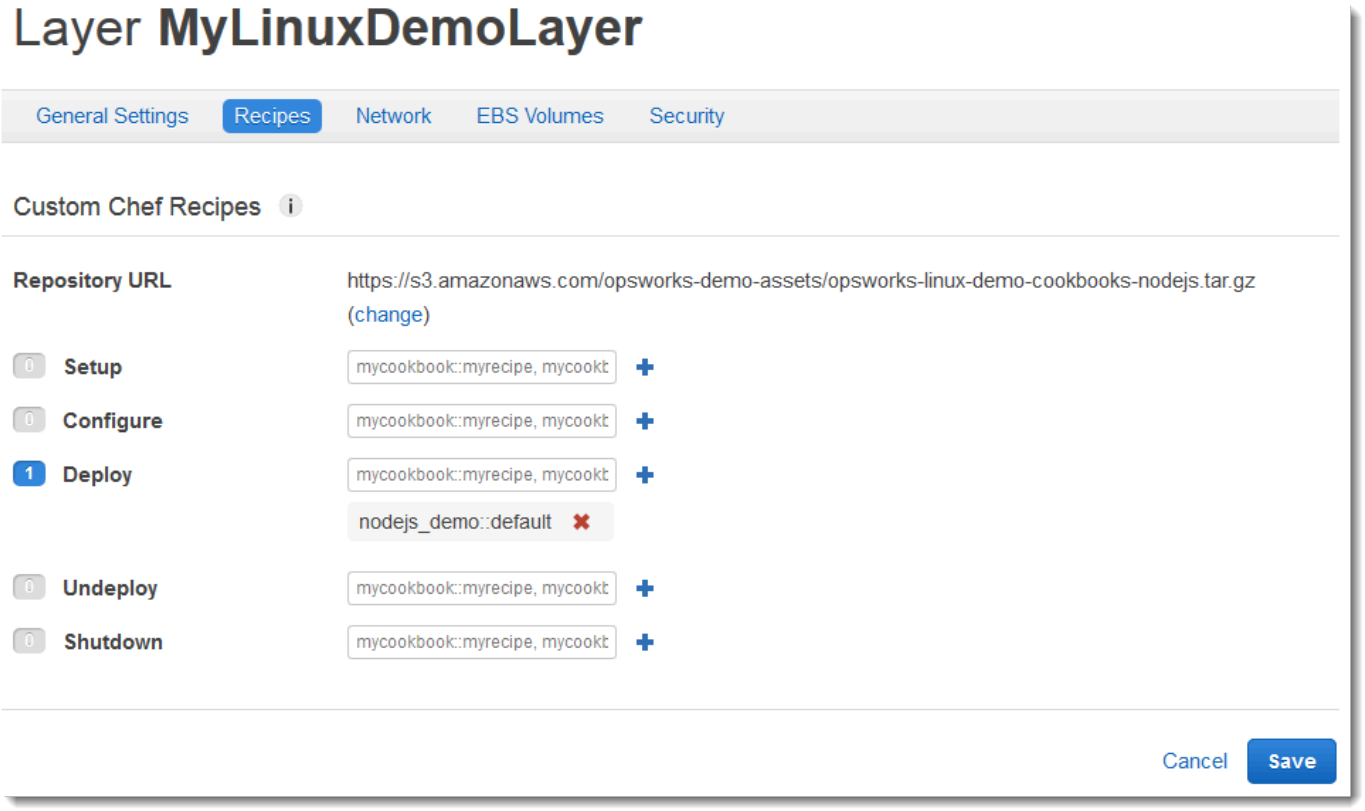
To specify the recipe to use to deploy the app to the instance

1. In the service navigation pane, choose **Layers**. The **Layers** page is displayed.
2. For **MyLinuxDemoLayer**, choose **Recipes**:



   The **Layer MyLinuxDemoLayer** page is displayed with the **Recipes** tab open.
3. For **Custom Chef Recipes**, for **Deploy**, type **`nodejs_demo::default`**, and then press **Enter**. `nodejs_demo` is the name of the cookbook and `default` is the name of the target recipe

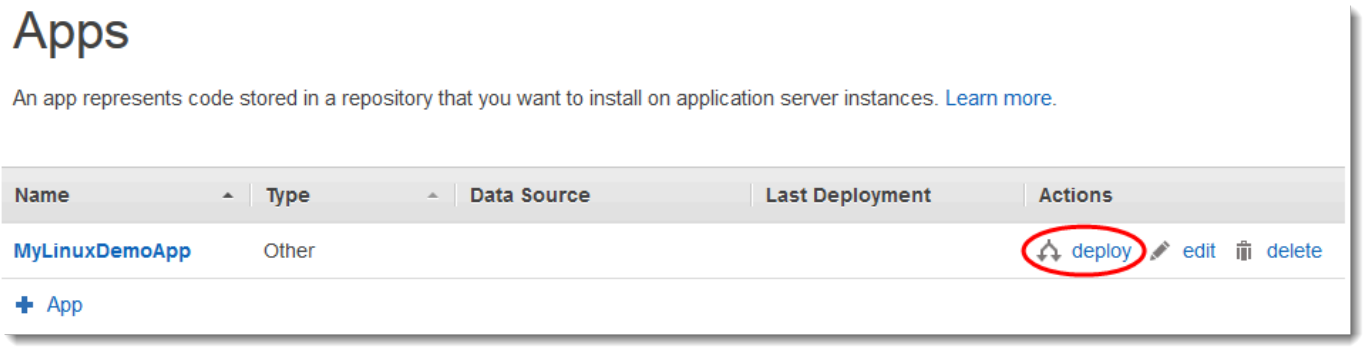
within the cookbook. (To explore the recipe's code, see Learning More: Explore the Cookbook Used in This Walkthrough.) Your results must match the following screenshot:



4. Choose **Save**. OpsWorks Stacks adds the recipe to the layer's Deploy lifecycle event.

To deploy the app to the instance

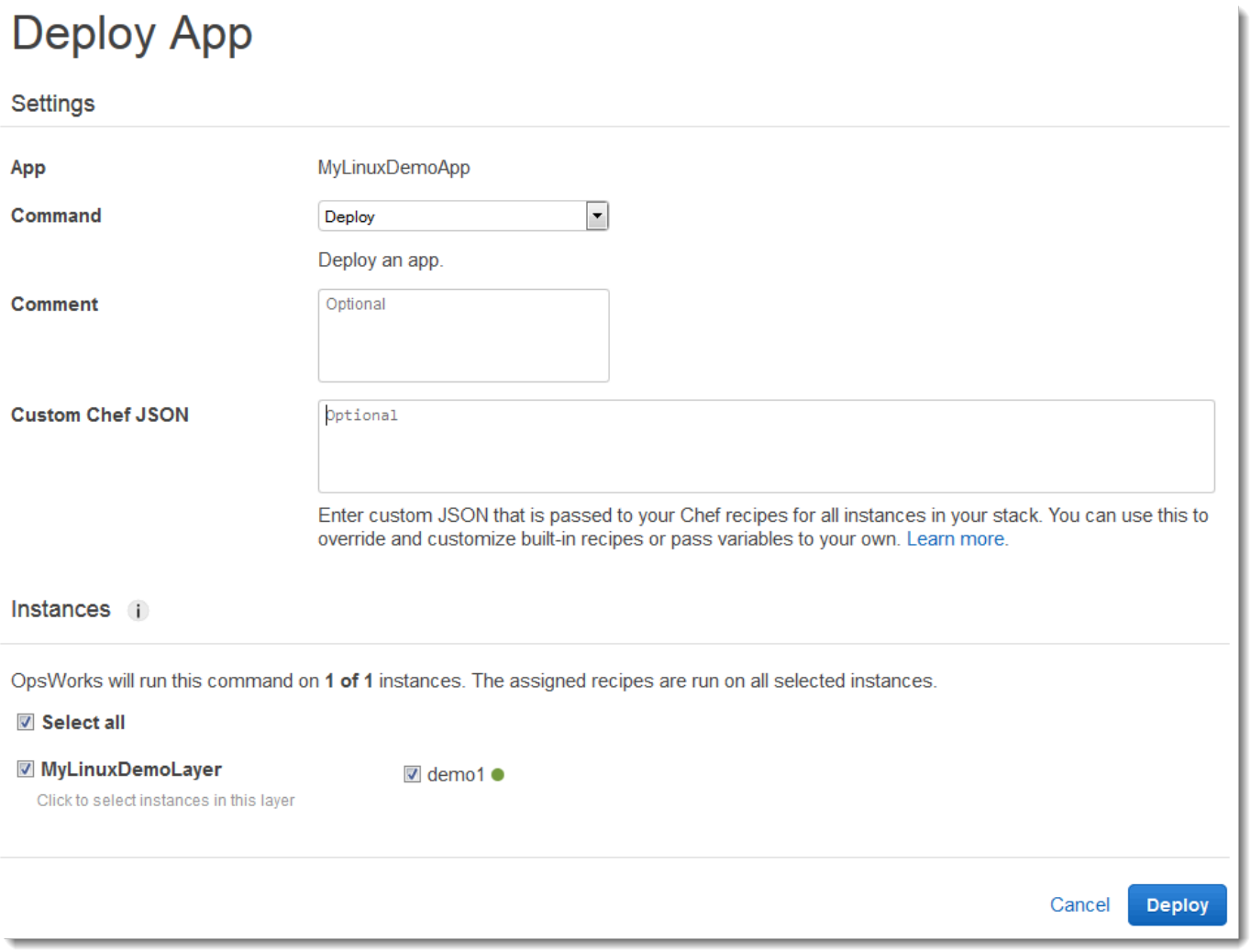
1. In the service navigation pane, choose **Apps**. The **Apps** page displays.
2. For **MyLinuxDemoApp**, for **Actions**, choose **deploy**, as displayed in the following screen shot:



3. On the **Deploy App** page, leave the defaults for the following:

   - **Command** (**Deploy**)
   - **Comment** (blank)
   - **Settings**, **Advanced**, **Custom Chef JSON** (blank)
   - **Instances**, **Advanced** (checked **Select all**, checked **MyLinuxDemoLayer**, checked **demo1**)

4. Your results must match the following screenshot:



5. Choose **Deploy**. The **Deployment MyLinuxDemoApp – deploy** page is displayed. **Status** changes from **running** to **successful**. A spinning circle displays next to **demo1**, which then changes to a green check mark. Note that this process can take several minutes. Do not proceed until you see both a **Status** of **successful** and the green check mark icon.

6. Your results must match the following screenshot except of course for **Created at**, **Completed at**, **Duration**, and **User**. If **status** is **failed**, then to troubleshoot, for **Log**, choose **show** to get details about the failure:



You have now successfully deployed the app to the instance.

In the next step, you will test the deployed app on the instance.

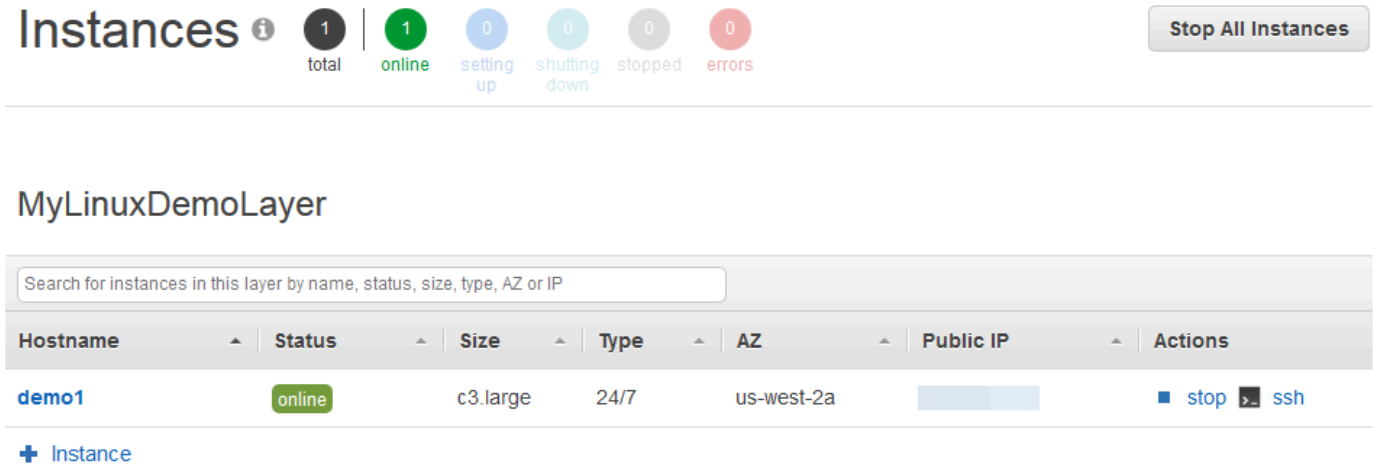
Step 7: Test the Deployed App on the Instance

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Now, test the app deployment on the instance.

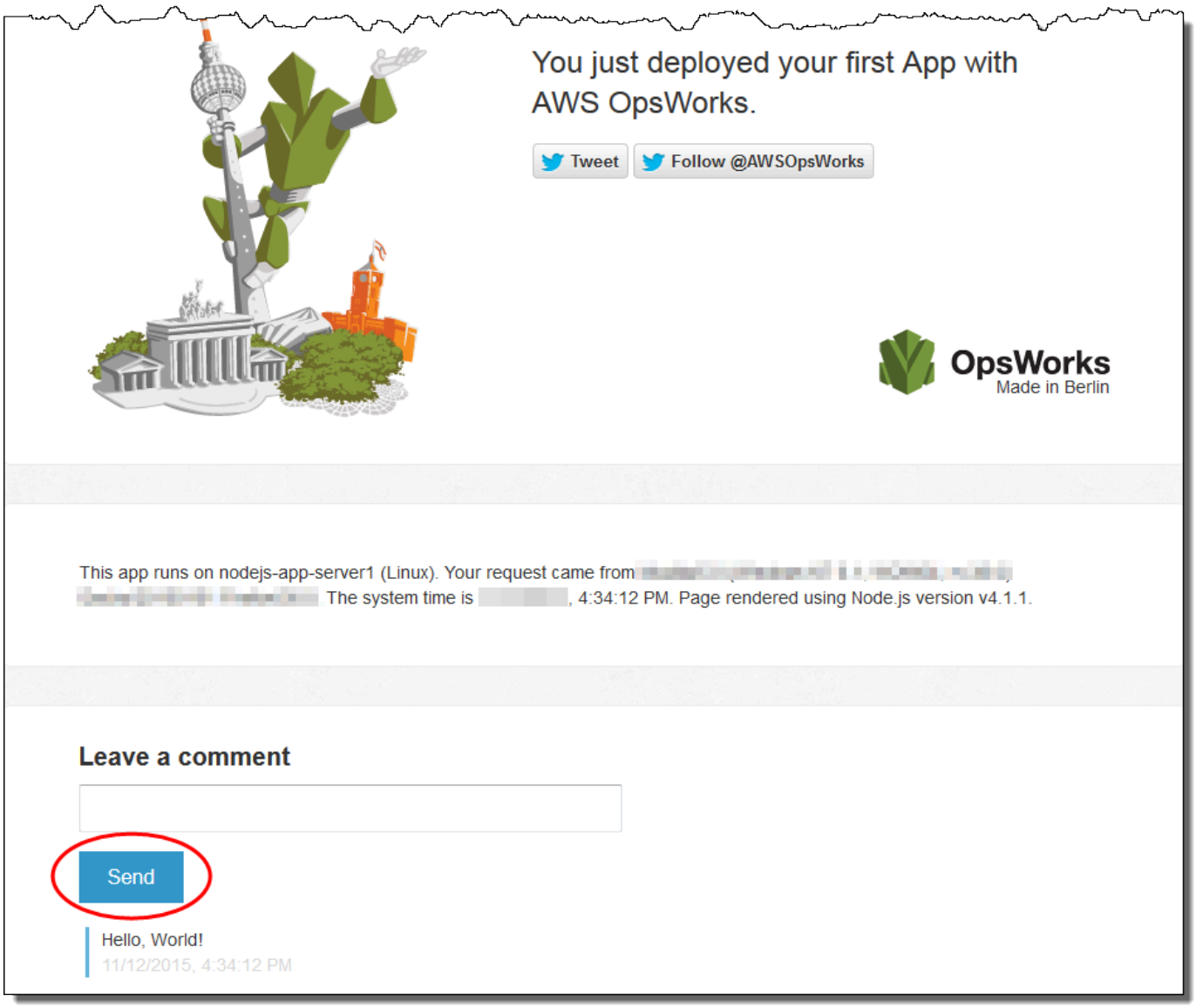
To test the deployment on the instance

1. In the service navigation pane, choose **Instances**. The **Instances** page is displayed.
2. For **MyLinuxDemoLayer**, for **demo1**, for **Public IP**, choose the IP address:



A new web browser tab displays the app.

3. On the congratulatory web page, in the **Leave a comment** text box, type a comment, and then choose **Send** to test the app. The app adds your comment to the web page. Continue leaving comments and choosing **Send** as often as you want:



4. If you have a Twitter account, choose **Tweet** or **Follow @AWSOpsWorks**, and follow the on-screen directions to tweet about the app or to follow @AWSOpsWorks.

You have now successfully tested the deployed app on the instance.

In the next step, you can clean up the AWS resources that you used for this walkthrough. This next step is optional. You may want to keep using these AWS resources as you continue to learn more about OpsWorks Stacks. However, keeping these AWS resources around may result in some ongoing charges to your AWS account. If you want to keep these AWS resources around for later use, you have now completed this walkthrough, and you can skip ahead to Next Steps.

Step 8 (Optional): Clean Up

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

To prevent incurring additional charges to your AWS account, you can delete the AWS resources that were used for this walkthrough. These AWS resources include the OpsWorks Stacks stack and the stack's components. (For more information, see OpsWorks Pricing.) However, you might want to keep using these AWS resources as you continue to learn more about OpsWorks Stacks. If you want to keep these AWS resources available, you have now completed this walkthrough, and you can skip to Next Steps.

Content stored in the resources that you created for this walkthrough can contain personally-identifying information. If you no longer want this information to be stored by AWS, follow steps in this topic.

To delete the app from the stack

1. In the OpsWorks Stacks console, in the service navigation pane, choose **Apps**. The **Apps** page is displayed.
2. For **MyLinuxDemoApp**, for **Actions**, choose **delete**. When the confirmation message is displayed, choose **Delete**. OpsWorks Stacks deletes the app.

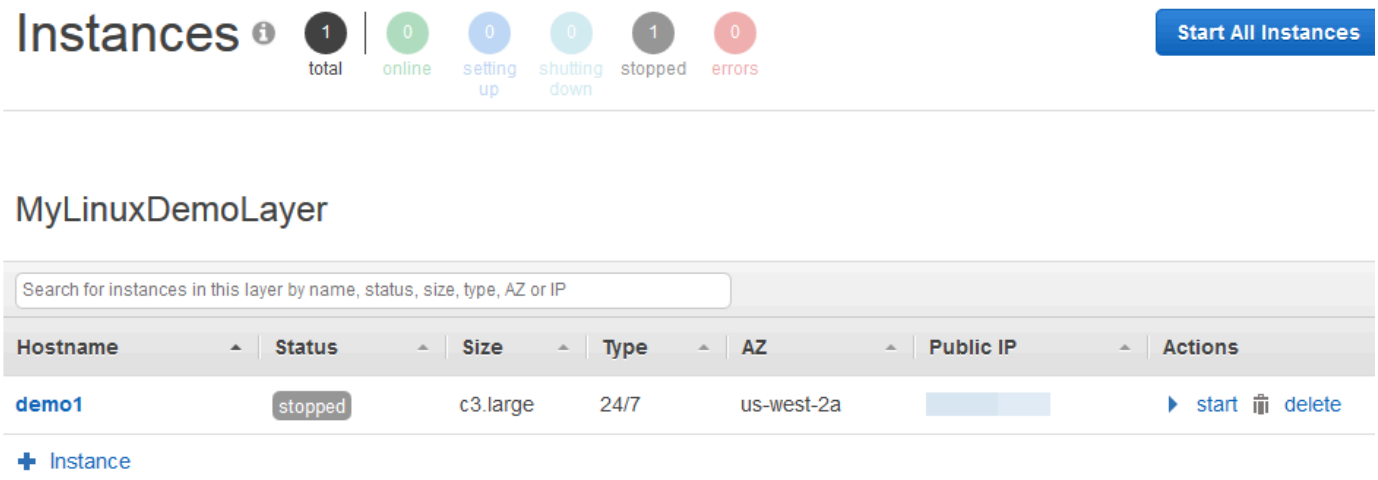
To delete the instance for the stack

1. In the service navigation pane, choose **Instances**. The **Instances** page is displayed.
2. For **MyLinuxDemoLayer**, for **demo1**, for **Actions**, choose **stop**. When you see the confirmation message, choose **Stop**. The following happens.

   - **Status** changes from **online** to **stopping**, and eventually to **stopped**.
   - **online** changes from **1** to **0**.
   - **shutting down** changes from **0** to **1**, and eventually back to **0**.

- **stopped** eventually changes from **0** to **1**.

This process can take a few minutes. When OpsWorks Stacks is finished, the following results are shown.



3. For **Actions**, choose **delete**. When you see the confirmation message, choose **Delete**. OpsWorks Stacks deletes the instance and displays the **No instances** message.

To delete the stack

1. In the service navigation pane, choose **Stack**. The **MyLinuxDemoStack** page is displayed.
2. Choose **Delete Stack**. When you see the confirmation message, choose **Delete**. OpsWorks Stacks deletes the stack and displays the **OpsWorks Dashboard** page.

Optionally, you can delete the user and Amazon EC2 key pair that you used for this walkthrough, if you don't want to reuse them for access to other AWS services and EC2 instances. For instructions, see Deleting an IAM user and Amazon EC2 key pairs and Linux instances.

You have now completed this walkthrough. For more information, see Next Steps.

Next Steps

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate

their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Now that you have completed this walkthrough, you can learn more about using OpsWorks Stacks:

- Explore the cookbook and the app that you used for this walkthrough. See Learning More: Explore the Cookbook Used in This Walkthrough and Learning More: Explore the App Used in This Walkthrough.
- Practice using OpsWorks Stacks with Windows instances. See Getting Started: Windows.
- Learn more about stacks by learning how to Create a New Stack.
- Learn more about layers by Editing an OpsWorks Layer's Configuration.
- Learn more about instances by Adding an Instance to a Layer.
- Learn more about apps by Deploying Apps.
- Learn more about Cookbooks and Recipes.
- Create your own cookbooks. See Getting Started: Cookbooks.
- Learn about controlling access to stacks with Security and Permissions.

Learning More: Explore the Cookbook Used in This Walkthrough

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This topic describes the cookbook that OpsWorks Stacks used for the walkthrough.

A *cookbook* is a Chef concept. Cookbooks are archive files that contain configuration information, such as recipes, attribute values, files, templates, libraries, definitions, and custom resources. A *recipe* is also a Chef concept. Recipes are instructions, written with Ruby language syntax,

that specify the resources to use and the order in which those resources are applied. For more information, go to About Cookbooks and About Recipes on the Learn Chef website.

To see the contents of the cookbook used in this walkthrough, extract the contents of the opsworks-linux-demo-cookbooks-nodejs.tar.gz file to an empty directory on your local workstation. (You can also log in to the instance that you deployed the cookbook to and explore the contents of the `/var/chef/cookbooks` directory.)

The `default.rb` file in the `cookbooks/nodejs_demo/recipes` directory is where the cookbook runs its code:

```
app = search(:aws_opsworks_app).first
app_path = "/srv/#{app['shortname']}"

package "git" do
  options "--force-yes" if node["platform"] == "ubuntu" && node["platform_version"] ==
 "18.04"
end

application app_path do
  javascript "4"
  environment.update("PORT" => "80")

  git app_path do
    repository app["app_source"]["url"]
    revision app["app_source"]["revision"]
  end

  link "#{app_path}/server.js" do
    to "#{app_path}/index.js"
  end

  npm_install
  npm_start
end
```

Here's what the file does:

- `search(:aws_opsworks_app).first` uses Chef search to look up information about the app that will eventually be deployed to the instance. This information includes settings such as the app's short name and its source repository details. Because only one app was deployed in this walkthrough, Chef search gets these settings from the first item of information within the

`aws_opsworks_app` search index on the instance. Whenever an instance is launched, OpsWorks Stacks stores this and other related information as a set of data bags on the instance itself, and you get the data bag contents through Chef search. Although you can hard code these settings into this recipe, using data bags and Chef search is a more robust approach. For more information about data bags, see the OpsWorks Stacks Data Bag Reference. See also About Data Bags on the Learn Chef website. For more information about Chef search, go to About Search on the Learn Chef website.

- The `package` resource installs Git on the instance.
- The `application` resource describes and deploys web applications:
  - `javascript` is the version of the JavaScript runtime to install.
  - `environment` sets an environment variable.
  - `git` gets the source code from the specified repository and branch.
  - `app_path` is the path to clone the repository to. If the path doesn't exist on the instance, OpsWorks Stacks creates it.
  - `link` creates a symbolic link.
  - `npm_install` installs Node Package Manager, the default package manager for Node.js.
  - `npm_start` runs Node.js.

Although OpsWorks Stacks created the cookbook used for this walkthrough, you can create your own cookbooks. To learn how, see Getting Started: Cookbooks. Also, go to About Cookbooks, About Recipes, and Learn the Chef Basics on Ubuntu on the Learn Chef website, and the "Our first Chef cookbook" section in First steps with Chef on the Getting started with Chef website.

Learning More: Explore the App Used in This Walkthrough

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This topic describes the app that OpsWorks Stacks deploys to the instance for this walkthrough.

To see the app's source code, extract the contents of the opsworks-windows-demo-nodejs GitHub repository to an empty directory on your local workstation. You can also log in to the instance that you deployed the cookbook to and explore the contents of the `/srv/mylinuxdemoapp` directory.

The `index.js` file contains the most significant code for the app:

```
var express = require('express');
var app = express();
var path = require('path');
var os = require('os');
var bodyParser = require('body-parser');
var fs = require('fs');

var add_comment = function(comment) {
  var comments = get_comments();
  comments.push({"date": new Date(), "text": comment});
  fs.writeFileSync('./comments.json', JSON.stringify(comments));
};

var get_comments = function() {
  var comments;
  if (fs.existsSync('./comments.json')) {
    comments = fs.readFileSync('./comments.json');
    comments = JSON.parse(comments);
  } else {
    comments = [];
  }
  return comments;
};

app.use(function log (req, res, next) {
  console.log([req.method, req.url].join(' '));
  next();
});
app.use(express.static('public'));
app.use(bodyParser.urlencoded({ extended: false }))

app.set('view engine', 'jade');
app.get('/', function(req, res) {
  var comments = get_comments();
  res.render("index",
    { agent: req.headers['user-agent'],
      hostname: os.hostname(),
```

```
      nodeversion: process.version,
      time: new Date(),
      admin: (process.env.APP_ADMIN_EMAIL || "admin@unconfigured-value.com" ),
      comments: get_comments()
    });
});

app.post('/', function(req, res) {
  var comment = req.body.comment;
  if (comment) {
    add_comment(comment);
    console.log("Got comment: " + comment);
  }
  res.redirect("/#form-section");
});

var server = app.listen(process.env.PORT || 3000, function() {
  console.log('Listening on %s', process.env.PORT);
});
```

Here's what the file does:

- `require` loads modules that contain some dependent code that this web app needs to run as expected.
- The `add_comment` and `get_comments` functions write information to, and read information from, the `comments.json` file.
- For information about `app.get`, `app.listen`, `app.post`, `app.set`, and `app.use`, see the Express API Reference.

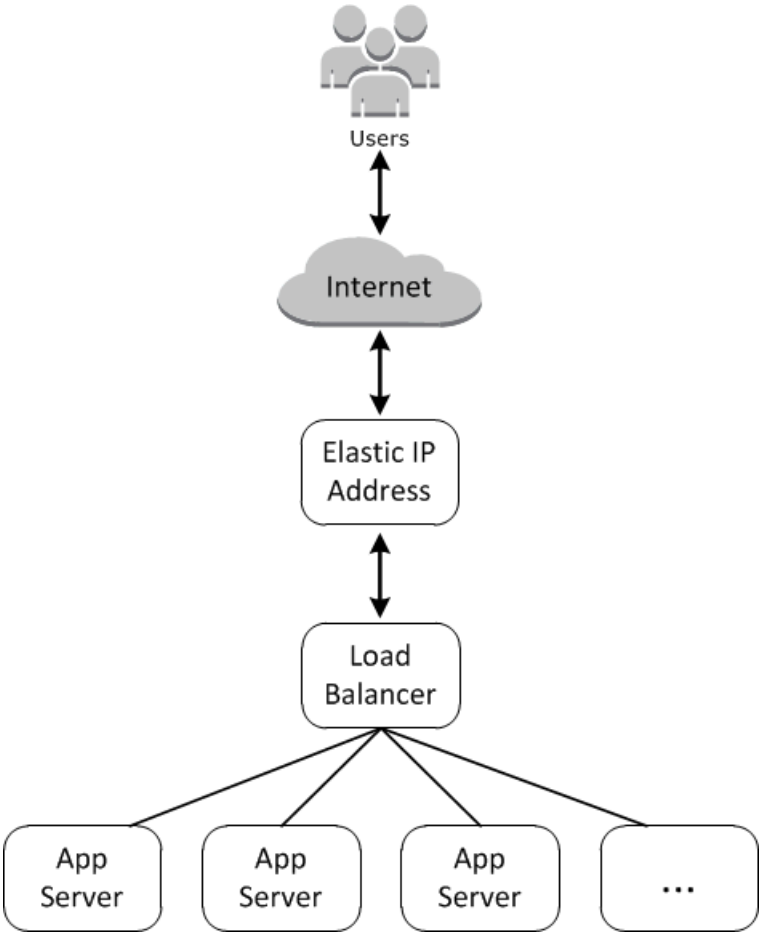
To learn how to create and package your app for deployment, see Application Source.

Getting Started with Windows Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Cloud-based applications usually require a group of related resources—application servers, database servers, and so on—that must be created and managed collectively. This collection of instances is called a *stack*. A simple application stack might look something like the following.



The basic architecture consists of the following:

- An Elastic IP address to receive user requests.
- A load balancer to distribute incoming requests evenly across the application servers.
- A set of application server instances, as many as needed to handle the traffic.

In addition, you typically need a way to distribute applications to the application servers, manage user permissions, and so on.

OpsWorks Stacks provides a simple and straightforward way to create and manage stacks and their associated applications and resources. This chapter introduces the basics of OpsWorks Stacks—along with some of its more sophisticated features—by walking you through the process of creating the application server stack in the diagram. It uses an incremental development model that OpsWorks Stacks makes easy to follow: Set up a basic stack and, after it's working correctly, add components until you arrive at a full-featured implementation.

- Step 1: Complete the Prerequisites shows how to get set up to start the walkthrough.
- Step 2: Create a Basic Application Server Stack shows how to create a basic stack to support Internet Information Services (IIS) and deploy an app to the server.
- Step 3: Scale Out IISExample shows how to scale out a stack to handle increased load by adding more application servers, a load balancer to distribute incoming traffic, and an Elastic IP address to receive incoming requests.

Topics

- Step 1: Complete the Prerequisites
- Step 2: Create a Basic Application Server Stack
- Step 3: Scale Out IISExample
- Next Steps

Step 1: Complete the Prerequisites

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Complete the following setup steps before you can start the walkthrough. These setup steps include signing up for an AWS account, creating an administrative user, and assigning access permissions to OpsWorks Stacks.

If you have already completed the Getting Started: Sample or Getting Started: Linux walkthroughs, then you have met the prerequisites for this walkthrough, and you can skip ahead to Step 2: Create a Basic Application Server Stack.

Topics

- Sign up for an AWS account
- Create a user with administrative access
- Assign Service Access Permissions
- Ensure OpsWorks Stacks Users are Added to Your Domain

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Assign Service Access Permissions

Enable access to the OpsWorks Stacks service (and related services that OpsWorks Stacks relies on) by adding the `AWSOpsWorks_FullAccess` and `AmazonS3FullAccess` permissions to your role or user.

For more information about adding permissions, see Adding IAM identity permissions (console).

Ensure OpsWorks Stacks Users are Added to Your Domain

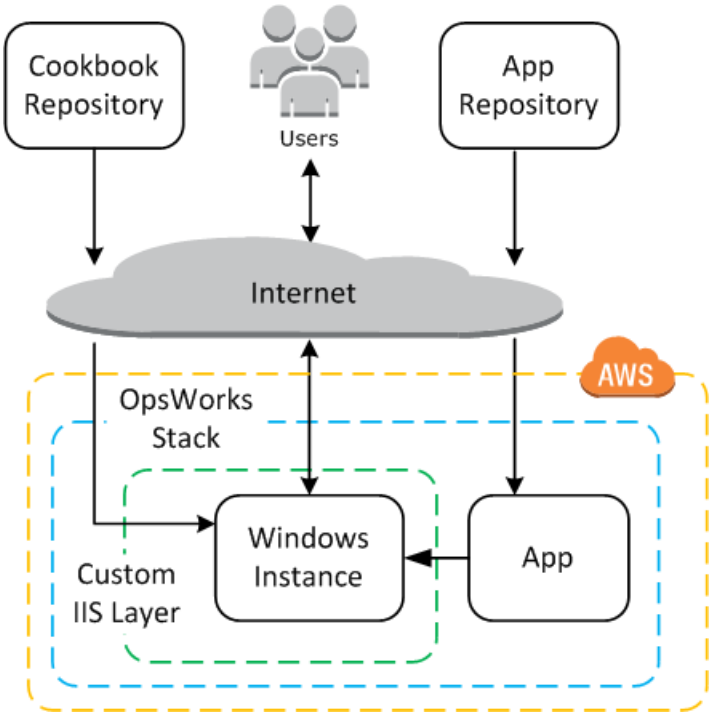
In a Chef 12.2 stack, the included `aws_opsworks_users` cookbook creates users that have SSH and Remote Desktop Protocol (RDP) access to Windows-based instances. When you join Windows instances in your stack to an Active Directory domain, this cookbook run can fail if the OpsWorks Stacks users do not exist in Active Directory. If the users are not recognized in Active Directory, instances can enter a `setup failed` state when you restart them after joining them to a domain. For domain-joined Windows instances, it is not sufficient to grant OpsWorks Stacks users SSH/RDP access on the user permissions page.

Before you join Windows instances in a Chef 12.2 stack to an Active Directory domain, be sure all OpsWorks Stacks users of the Windows-based stack are members of the domain. The best way to do this is to configure federated identity with IAM before creating your Windows-based stack, and then import federated users into OpsWorks Stacks before joining instances in your stack to a domain. For more information about how to do this, see Enabling Federation to AWS Using Windows Active Directory, ADFS, and SAML 2.0 in the AWS Security Blog, and Federating Existing Users in the *IAM User Guide*.

Step 2: Create a Basic Application Server Stack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

A basic application server stack consists of a single application server instance with a public IP address to receive user requests. Application code and any related files are stored in a separate repository and deployed from there to the server. The following diagram illustrates such a stack.



The stack has the following components:

- A *layer*, which represents a group of instances and specifies how they are to be configured.

  The layer in this example represents a group of IIS instances.
- An *instance*, which represents an Amazon EC2 instance.

  In this case, the layer configures a single instance to run IIS, but layers can have any number of instances.
- An *app*, which contains the information required to install an application on the instance.
- A *cookbook*, which contains custom Chef recipes that support the custom IIS layer. The cookbook and app code are stored in remote repositories, such as an archive file in an Amazon S3 bucket or a Git repository.

The following sections describe how to use the OpsWorks Stacks console to create the stack and deploy the application.

Topics

- Step 2.1: Create the Stack
- Step 2.2: Authorize RDP Access
- Step 2.3: Implement a Custom Cookbook
- Step 2.4: Add an IIS Layer
- Step 2.5: Deploy an App

Step 2.1: Create the Stack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You start an OpsWorks Stacks project by creating a stack, which acts as a container for your instances and other resources. The stack configuration specifies some basic settings, such as the AWS region and the default operating system, that are shared by all the stack's instances.

To create a new stack

1. **Add a Stack**

   If you haven't done so already, sign in to the OpsWorks Stacks console.

   - If the account has no existing stacks, you will see the **Welcome to AWS OpsWorks** page; choose **Add your first stack**.
   - Otherwise, you will see the OpsWorks Stacks dashboard, which lists your account's stacks; choose **Add Stack**.

2. **Configure the Stack**

   On the **Add Stack** page, choose **Chef 12 stack** and specify the following settings:

Stack name

Enter a name for your stack, which can contain alphanumeric characters (a–z, A–Z, and 0–9), and hyphens (-). The example stack for this walkthrough is named **`IISWalkthrough`**.

Region

Select US West (Oregon) as the stack's region.

You can create a stack in any region, but we recommend US West (Oregon) for tutorials.

Default operating system

Choose **Windows**, and then specify **Microsoft Windows Server 2022 Base**, which is the default setting.

Use custom Chef cookbooks

For the purposes of this walkthrough, specify **No** for this option.

3. Choose **Advanced** to confirm that you have an IAM role and the default IAM instance profile selected.

   IAM role

   Specify the stack's IAM (AWS Identity and Access Management) role. OpsWorks Stacks needs to access other AWS services to perform tasks such as creating and managing Amazon EC2 instances. **IAM role** specifies the role that OpsWorks Stacks assumes to work with other AWS services on your behalf. For more information, see Allowing OpsWorks Stacks to Act on Your Behalf.

   - If your account has an existing OpsWorks Stacks IAM role, you can select it from the list.

     If the role was created by OpsWorks Stacks, it will be named `aws-opsworks-service-role`.

   - Otherwise, select **New IAM Role** to direct OpsWorks Stacks to create a new role for you with the correct permissions.

     Note: If you have OpsWorks Stacks Full Access permissions, creating a new role requires several additional IAM permissions. For more information, see Example Policies.

4. Accept the default values for the other settings and choose **Add Stack**. For more information on the various stack settings, see Create a New Stack.

Step 2.2: Authorize RDP Access

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Now that you have created a stack, you will create a layer and add a Windows instance to the layer. However, before you can do so, you must configure the stack to allow you to use RDP to connect to the custom layer's instances. To do so, you must do the following:

- Add an inbound rule to the security group that controls RDP access.
- Set your OpsWorks Stacks permissions for this stack to allow RDP access.

When you create the first stack in a region, OpsWorks Stacks creates a set of security groups. They include one named something like `AWS-OpsWorks-RDP-Server`, which OpsWorks Stacks attaches to all Windows instances to allow RDP access. However, by default, this security group does not have any rules, so you must add an inbound rule to allow RDP access to your instances.

To allow RDP access

1. Open the Amazon EC2 console, set it to the stack's region, and choose **Security Groups** from the navigation pane.
2. Choose **AWS-OpsWorks-RDP-Server**, choose the **Inbound** tab, and choose **Edit**.
3. Choose **Add Rule** and specify the following settings:

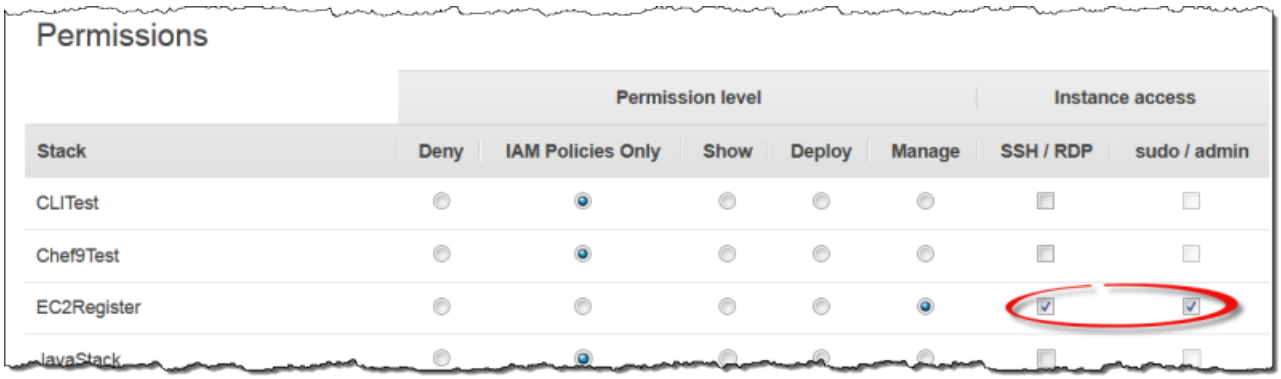
   - **Type** – **RDP**.
   - **Source** – The permissible source IP addresses.

     You typically allow inbound RDP requests from your IP address or a specified IP address range (typically your corporate IP address range). For learning purposes, it's often sufficient to specify 0.0.0.0/0, which allows RDP access from any IP address.

The security group allows the instance to receive RDP connection requests, but that's only half the story. An ordinary user will log into the instance using a password provided by OpsWorks Stacks. To have OpsWorks Stacks generate that password, you must explicitly authorize RDP access for the user.

To authorize RDP for a user

1. In the OpsWorks Stacks dashboard, choose the **IISWalkthrough** stack.
2. In the navigation pane for the stack, choose **Permissions**.
3. On the Permissions page, choose **Edit**.
4. In the list of users, select the checkbox for **SSH/RDP** for the user to whom you want to grant necessary permissions. If you want the user to also have administrator permissions, select **sudo/admin** as well.



5. Choose **Save**.

The user can then get a password and use it to log in to the instance, as described later.

> **Note**
>
> You also can log in as Administrator. For more information, see Logging in As Administrator.

Step 2.3: Implement a Custom Cookbook

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate

their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Although a stack is basically a container for instances, you don't add instances directly to a stack. You add one or more layers, each of which represents a group of related instances, and then add instances to the layers.

A layer is basically a blueprint that OpsWorks Stacks uses to create a set of Amazon EC2 instances with the same configuration. An instance starts with a base version of the operating system, and the instance's layer performs a variety of tasks on the instance to implement that blueprint, which can include:

- Creating directories and files
- Managing users
- Installing and configuring software
- Starting or stopping servers
- Deploying application code and related files.

A layer performs tasks on instances by running Chef recipes—recipes for short. A recipe is a Ruby application that uses Chef's domain-specific language (DSL) to describe the final state of the instance. With OpsWorks Stacks, each recipe is usually assigned to one of the layer's lifecycle events: Setup, Configuration, Deploy, Undeploy, and Shutdown. When a lifecycle event occurs on an instance, OpsWorks Stacks runs the event's recipes to perform the appropriate tasks. For example, the Setup event occurs after an instance finishes booting. OpsWorks Stacks then runs the Setup recipes, which typically perform tasks such as installing and configuring server software and starting related services.

OpsWorks Stacks provides each layer with a set of built-in recipes that perform standard tasks. You can extend a layer's functionality by implementing custom recipes to perform additional tasks and assigning them to the layer's lifecycle events. Windows stacks support custom layers, which have a minimal set of recipes that perform only a few basic tasks. To add functionality to your Windows instances, you must implement custom recipes to install software, deploy applications, and so on. This topic describes how to create a simple custom layer to support IIS instances.

Topics

- A Quick Introduction to Cookbooks and Recipes
- Implement a Recipe to Install and Start IIS
- Enable the Custom Cookbook

A Quick Introduction to Cookbooks and Recipes

A recipe defines one or more aspects of an instance's expected state: what directories it should have, what software packages should be installed, what apps should be deployed, and so on. Recipes are packaged in a *cookbook*, which typically contains one or more related recipes, plus associated files such as templates for creating configuration files.

This topic is a very basic introduction to recipes, just enough to show you how to implement a cookbook to support a simple custom IIS layer. For a more general introduction to cookbooks, see Cookbooks and Recipes. For a detailed tutorial introduction to implementing cookbooks, including some Windows-specific topics, see Cookbooks 101.

Chef recipes are technically Ruby applications, but most, if not all, of the code is in the Chef DSL. The DSL consists largely of a set of *resources*, which you can use to declaratively specify an aspect of the instances state. For example, a `directory` resource defines a directory to be added to the system. The following example defines a `C:\data` directory with full-control rights that belongs to the specified user and does not inherit rights from the parent directory.

```
directory 'C:\data' do
  rights :full_control, 'WORKGROUP\username'
  inherits false
  action :create
end
```

When Chef runs a recipe, it executes each resource by passing the data to an associated *provider*, a Ruby object that handles the details of modifying the instance state. In this case, the provider creates a new directory with the specified configuration.

The custom cookbook for the custom IIS layer must perform the following tasks:

- Install the IIS feature and start the service.

  You typically perform this task during setup, right after the instance finished booting.
- Deploy an app to the instance, a simple HTML page for this example.

You typically perform this task during setup. However, apps usually need to be updated regularly, so you also need to deploy updates while the instance is online.

You could have a single recipe perform all of these tasks. However, the preferred approach is to have separate recipes for setup and deployment tasks. That way, you can deploy app updates at any time without also running setup code. The following describes how to set up a cookbook to support a custom IIS layer. Subsequent topics will show how to implement the recipes.

To get started

1. Create a directory named `iis-cookbook` in a convenient location on your workstation.
2. Add a `metadata.rb` file with the following content to `iis-cookbook`.

   ```
   name "iis-cookbook"
   version "0.1.0"
   ```

   This example uses a minimal `metadata.rb`. For more information on how you can use this file, see metadata.rb.
3. Add a `recipes` directory to `iis-cookbook`.

   This directory, which must be named `recipes`, contains the cookbook's recipes.

In general, cookbooks can contain a variety of other directories. For example, if a recipe uses a template to create a configuration file, the template usually goes in the `templates\default` directory. The cookbook for this example consists entirely of recipes, so it needs no other directories. Also, this example uses a single cookbook, but you can use as many as you need; multiple cookbooks are often preferable for complex projects. For example, you could have separate cookbooks for setup and deployment tasks. For more cookbook examples, see Cookbooks and Recipes.

Implement a Recipe to Install and Start IIS

IIS is a Windows *feature*, one of a set of optional system components that you can install on Windows Server. You can have a recipe install IIS in either of the following ways:

- By using a `powershell_script` resource to run the `Install-WindowsFeature` cmdlet.
- By using the Chef windows cookbook `windows_feature` resource.

The `windows` cookbook contains a set of resources whose providers use Deployment Image Servicing and Management (DISM) to perform a variety of tasks on Windows instances, including feature installation.

> **Note**
>
> `powershell_script` is among the most useful resources for Windows recipes. You can use it to perform a variety of tasks on an instance by running a PowerShell script or cmdlet. It's especially useful for tasks that aren't supported by a Chef resource.

This example runs a PowerShell script to install and start Web Server (IIS). The `windows` cookbook is described later. For an example of how to use `windows_feature` to install IIS, see Installing a Windows Feature: IIS.

Add a recipe named `install.rb` with the following contents to the cookbook's `recipes` directory.

```
powershell_script 'Install IIS' do
  code 'Install-WindowsFeature Web-Server'
  not_if "(Get-WindowsFeature -Name Web-Server).Installed"
end

service 'w3svc' do
  action [:start, :enable]
end
```

The recipe contains two resources.

powershell_script

`powershell_script` runs the specified PowerShell script or cmdlet. The example has the following attribute settings:

- `code` – The PowerShell cmdlets to run.

  This example runs an `Install-WindowsFeature` cmdlet, which installs Web Server (IIS). In general, the `code` attribute can have any number of lines, so you can run as many cmdlets as you need.

- `not-if` – A *guard attribute* that ensures that the recipe installs IIS only if it has not yet been installed.

  You generally want recipes to be *idempotent*, so they do not waste time performing the same task more than once.

Every resource has an action, which specifies the action the provider is to take. There is no explicit action for this example, so the provider takes the default `:run` action, which runs the specified PowerShell script. For more information, see Running a Windows PowerShell Script.

service

A `service` manages a service, the Web Server IIS service (W3SVC) in this case. The example uses default attributes and specifies two actions, `:start` and `:enable`, which start and enable IIS.

> **Note**
>
> If you want to install software that uses a package installer, such as MSI, you can use a `windows_package` resource. For more information, see Installing a Package.

Enable the Custom Cookbook

OpsWorks Stacks runs recipes from a local cache on each instance. To run your custom recipes, you must do the following:

- Store the cookbook in a remote repository.

  OpsWorks Stacks downloads the cookbooks from this repository to each instance's local cache.
- Edit the stack to enable custom cookbooks.

  Custom cookbooks are disabled by default, so you must enable custom cookbooks for the stack and provide the repository URL and related information.

OpsWorks Stacks supports S3 archives and Git repositories for custom cookbooks; this example uses an S3 archive. For more information, see Cookbook Repositories.

To use an S3 archive

1. Create a `.zip` archive of the `iis-cookbook` directory.

   OpsWorks Stacks also supports `.tgz` (gzip compressed tar) archives for Windows stacks.

2. Upload the archive to an S3 bucket in the US West (N. California) region, and make the file public. You can also use private S3 archives, but public archives are sufficient for this example and somewhat simpler to work with.

   a. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.

   b. If you do not already have a bucket in `us-west-1`, choose **Create Bucket** and create a bucket in the US West (N. California) region.

   c. In the buckets list, choose the name of bucket to which you want to upload the file, and then choose **Upload**.

   d. Choose **Add Files**.

   e. Select the archive file to upload, and then choose **Open**.

   f. At the bottom of the **Upload - Select Files and Folders** dialog, choose **Set Details**.

   g. At the bottom of the **Set Details** dialog, choose **Set Permissions**.

   h. In the **Set Permissions** dialog, choose **Make everything public**.

   i. At the bottom of the **Set Permissions** dialog, choose **Start Upload**. When the upload finishes, the `iis-cookbook.zip` file appears in your bucket.

   j. Choose the bucket, and then choose the **Properties** tab for the bucket. Next to **Link**, record the archive file's URL for later use.

   For more information about uploading files to an Amazon S3 bucket, see How Do I Upload Files and Folders to an S3 Bucket? in the *Amazon S3 Console User Guide*.

> **Important**
>
> Up to this point, the walkthrough has cost you only a little time; the OpsWorks Stacks service itself is free. However, you must pay for any AWS resources that you use, such as Amazon S3 storage. As soon as you upload the archive you begin incurring charges. For more information, see AWS Pricing.

To enable custom cookbooks for the stack

1. In the OpsWorks Stacks console, choose **Stack** in the navigation pane, and then choose **Stack Settings** on the upper right.
2. On the upper right of the **Settings** page, choose **Edit**.
3. On the **Settings** page, set **Use custom Chef cookbooks** to **Yes** and enter the following information:
   - Repository type – **S3 Archive**.
   - Repository URL – The S3 URL of the cookbook archive file that you recorded earlier.
4. Choose **Save** to update the stack configuration.

OpsWorks Stacks installs your custom cookbook on all new instances. Note that OpsWorks Stacks does not automatically install or update custom cookbooks on online instances. You can do that manually, as described later.

Step 2.4: Add an IIS Layer

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Your cookbook has one recipe that just installs and starts IIS. This is enough to create the layer and verify that you have a working IIS instance. Later, you will add application deployment functionality to the layer.

Create a Layer

You start by adding a layer to the stack. You then add functionality to that layer by assigning custom recipes to the appropriate lifecycle events.

To add an IIS layer to the stack

1. Choose **Layers** in the navigation pane and then choose **Add a layer**.

2. Configure the layer as follows:

   - **Name**– **`IISExample`**
   - **Short name** – **`iisexample`**

     OpsWorks Stacks uses the short name to identify the layer internally. You also use the short name to identify the layer in recipes, although this example does not do so. You can specify any short name, but it can consist only of lowercase alphanumeric characters and a small number of punctuation marks. For more information, see Custom Layers.

3. Choose **Add Layer**.

If you were to add an instance to IISWalkthrough at this point and start it, OpsWorks Stacks would automatically install the cookbooks but it would not run `install.rb`. After an instance is online, you can run recipes manually by using the Execute Recipes stack command. However, a better approach is to assign the recipe to one of the layer's lifecycle events. OpsWorks Stacks then automatically runs the recipe at the appropriate point in the instance's lifecycle.

Install and start IIS as soon as the instance finishes booting. To do this, assign `install.rb` to the layer's `Setup` event.

To assign the recipe to a lifecycle event

1. Choose **Layers** in the navigation pane
2. In the box for the **IISExample** layer, choose **Recipes**.
3. On the upper right, choose **Edit**.
4. Under **Custom Chef Recipes**, in the **Setup** recipes box, type **`iis-cookbook::install`**.

   > **Note**
   >
   > Use *`cookbook-name`*`::`*`recipe-name`* to identify recipes, where you omit the recipe name's `.rb` suffix.

5. Choose **+** to add the recipe to the layer. A red x appears next to the recipe to make it easy to remove later.
6. Choose **Save** to save the new configuration. The custom Setup recipes should now include `iis-cookbook::install`.

Add an Instance to the Layer and Start It

You can try the recipe out by adding an instance to the layer and starting the instance. OpsWorks Stacks automatically installs the cookbooks and runs `install.rb` during setup, as soon as the instance finishes booting.

To add an instance to a layer and start it

1. In the OpsWorks Stacks navigation pane, choose **Instances**.
2. Under **IISExample** layer, choose **Add an instance**.
3. Select the appropriate size. **t2.micro** (or the smallest size available to you) should be sufficient for the example.
4. Choose **Add Instance**. By default, OpsWorks Stacks generates instance names by appending an integer to the layer's short name, so the instance should be named **iisexample1**.
5. Choose **start** in the instance's **Actions** column to start the instance. OpsWorks Stacks will then launch an EC2 instance and run the Setup recipes to configure it. If the layer had any Deploy recipes at this point, OpsWorks Stacks would run them after the Setup recipes have finished.

   The process may take a number of minutes, during which the **Status** column displays a series of status states. When you get to **online** status, the setup process is complete and the instance is ready for use.

Confirm that IIS is Installed and Running

You can use RDP to connect to the instance and verify that your Setup recipe worked correctly.

To verify that IIS is installed and running

1. Choose **Instances** in the navigation pane and choose **rdp** in the **iisexample1** instance's **Actions** column. OpsWorks Stacks automatically generates an RDP password for you that expires after a specified time period.
2. Set **Session valid for** to 2 hours and choose **Generate Password**.
3. OpsWorks Stacks displays the password and also, for your convenience, the instance's public DNS name and user name. Copy all three and click **Acknowledge and close**.
4. Open your RDP client and use the data from Step 3 to connect to the instance.
5. On the instance, open Windows Explorer and examine the `C:` drive. It should have a `C:\inetpub` directory, which was created by the IIS installation.

6. Open the Control Panel **Administrative Tools** application, and then open **Services**. You should see the IIS service near the bottom of the list. It is named World Wide Web Publishing Service, and the status should be **running**.
7. Return to the OpsWorks Stacks console and choose the **iisexample1** instance's public IP address. Be sure you do this in OpsWorks Stacks, and not in the Amazon EC2 console. This automatically sends an HTTP request to the address, which should open the default IIS Welcome page.

The next topic discusses how to deploy an app to the instance, a simple static HTML page for this example. However, if you would like to take a break, choose **stop** in the **iisexample1** instance's **Actions** column to stop the instance and avoid incurring unnecessary charges. You can restart the instance when you are ready to continue.

Step 2.5: Deploy an App

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The IIS installation creates an `C:\inetpub\wwwroot` directory for your application's code and related files. The next step is to install an app in that directory. For this example, you will install a static HTML home page, `default.html`, in `C:\inetpub\wwwroot`. You can easily extend the general approach to handle more complex scenarios, such as ASP.NET applications.

You could include the application's files in your cookbook and have `install.rb` copy them to `C:\inetpub\wwwroot`. For examples of how to do this, see Example 6: Creating Files. However, this approach is not very flexible or efficient, and it is usually better to separate cookbook development from application development.

The preferred solution is to implement a separate deployment recipe that retrieves the application's code and related files from a repository—any repository you prefer, not just the cookbook repository—and installs it on each IIS server instance. This approach separates cookbook

development from application development and, when you need to update your app, it allows you to just run the deployment recipe again without having to update your cookbooks.

This topic shows how to implement a simple deployment recipe that deploys `default.htm` to your IIS server. You can readily extend this example to more complex applications.

Topics

- Create the Application and Store It in a Repository
- Implement a Recipe to Deploy the Application
- Update the Instance's Cookbooks
- Add the Recipe to the Custom IIS Layer
- Add an App
- Deploy the App and Open the Application

Create the Application and Store It in a Repository

You can use any repository you prefer for your applications. For simplicity, this example stores `default.htm` in a public S3 bucket.

To create the application

1. Create a directory named `iis-application` in a convenient location on your workstation.
2. Add a `default.htm` file to `iis-application` with the following content.

```
<!DOCTYPE html>
<html>
  <head>
    <title>IIS Example</title>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```

3. Create an S3 bucket, upload `default.htm` to the bucket, and record the URL for later use. For simplicity, make the file public.

Note

This is a very simple application, but you can extend the basic principles to handle production-level applications.

- For more complex applications with multiple files, it is usually simpler to create a .zip archive of `iis-application` and upload it to your S3 bucket.

  You can then download the .zip file and extract the contents to the appropriate directory. There's no need to download multiple files, create a directory structure, or so on.
- For a production application, you will probably want to keep your files private. For an example of how to have a recipe download files from a private S3 bucket, see Using the SDK for Ruby on an OpsWorks Stacks Windows Instance.
- You can store your application in any suitable repository.

  You typically download the application by using a repository's public API. This example uses the Amazon S3 API. If, for example, you store your application on GitHub, you can use the GitHub API.

Implement a Recipe to Deploy the Application

Add a recipe named `deploy.rb` to the `iis-cookbook` `recipes` directory, with the following contents.

```
chef_gem "aws-sdk-s3" do
  compile_time false
  action :install
end

ruby_block "download-object" do
  block do
    require 'aws-sdk-s3'

    #1
    # Aws.config[:ssl_ca_bundle] = 'C:\ProgramData\Git\bin\curl-ca-bundle.crt'
    Aws.use_bundled_cert!
```

```
    #2
    query = Chef::Search::Query.new
    app = query.search(:aws_opsworks_app, "type:other").first
    s3region = app[0][:environment][:S3REGION]
    s3bucket = app[0][:environment][:BUCKET]
    s3filename = app[0][:environment][:FILENAME]

    #3
    s3_client = Aws::S3::Client.new(region: s3region)
    s3_client.get_object(bucket: s3bucket,
                         key: s3filename,
                         response_target: 'C:\inetpub\wwwroot\default.htm')
  end
  action :run
end
```

This example uses SDK for Ruby v2 to download the file. However, OpsWorks Stacks does not install this SDK on Windows instances, so the recipe starts with `chef_gem` resource, which handles that task.

> **Note**
>
> The `chef_gem` resource installs gems into Chef's dedicated Ruby version, which is the version that recipes use. If you want to install a gem for a system-wide Ruby version, use the gem_package resource.

The bulk of the recipe is a `ruby_block` resource, which runs a block of Ruby code that uses the SDK for Ruby to download `default.htm`. The code in the `ruby_block` can be divided into the following sections, which correspond to the numbered comments in the code example.

1: Specify a Certificate Bundle

Amazon S3 uses SSL, so you need an appropriate certificate to download objects from an S3 bucket. SDK for Ruby v2 does not include a certificate bundle, so you must provide one and configure the SDK for Ruby to use it. OpsWorks Stacks does not install a certificate bundle directly, but it does install Git, which includes a certificate bundle (`curl-ca-bundle.crt`). For convenience, this example configures the SDK for Ruby to use the Git certificate bundle for SSL. You also can install your own bundle and configure the SDK accordingly.

2: Retrieve the Repository Data

To download an object from Amazon S3, you need the AWS region, bucket name, and key name. As described later, this example provides this information by associating a set of environment variables with the app. When you deploy an app, OpsWorks Stacks adds a set of attributes to the instance's node object. These attributes are essentially a hash table that contains the app configuration, including the environment variables. The app attributes for this application will look something like the following, in JSON format.

```
{
  "app_id": "8f71a9b5-de7f-451c-8505-3f35086e5bb3",
  "app_source": {
      "password": null,
      "revision": null,
      "ssh_key": null,
      "type": "other",
      "url": null,
      "user": null
  },
  "attributes": {
      "auto_bundle_on_deploy": true,
      "aws_flow_ruby_settings": {},
      "document_root": null,
      "rails_env": null
  },
  "data_sources": [{"type": "None"}],
  "domains": ["iis_example_app"],
  "enable_ssl": false,
  "environment": {
      "S3REGION": "us-west-2",
      "BUCKET": "windows-example-app",
      "FILENAME": "default.htm"
  },
  "name": "IIS-Example-App",
  "shortname": "iis_example_app",
  "ssl_configuration": {
      "certificate": null,
      "private_key": null,
      "chain": null
  },
  "type": "other",
  "deploy": true
```

```
}
```

The app's environment variables are stored in the `[:environment]` attribute. To retrieve them, use a Chef search query to retrieve the app's hash table, which is under the `aws_opsworks_app` node. This app will be defined as the `other` type, so the query searches for apps of that type. The recipe takes advantage of the fact that there is only one app on this instance, so the hash table of interest is just `app[0]`. For convenience, the recipe then assigns the region, bucket, and file names to variables.

For more information about how to use Chef search, see .Obtaining Attribute Values with Chef Search

3: Download the file

The third part of the recipe creates an S3 client object and uses its `get_object` method to download `default.htm` to the instance's `C:\inetpub\wwwroot` directory.

> **Note**
>
> A recipe is a Ruby application, so Ruby code doesn't necessarily have to be in a `ruby_block`. However, the code in the body of the recipe runs first, followed by the resources, in order. For this example, if you put the download code in the recipe body, it would fail because the `chef_gem` resource wouldn't have installed the SDK for Ruby yet. The code in the `ruby_block` resource executes when the resource executes, after the `chef_gem` resource has installed the SDK for Ruby.

Update the Instance's Cookbooks

OpsWorks Stacks automatically installs custom cookbooks on new instances. However, you are working with an existing instance, so you must update your cookbook manually.

To update the instance's cookbooks

1. Create a `.zip` archive of `iis-cookbook`, and upload it to the S3 bucket.

   This overwrites the existing cookbook, but the URL stays the same, so you don't need to update the stack configuration.
2. If your instance is not online, restart it.

3. After the instance is online, choose **Stack** in the navigation pane, and then choose **Run Command**.
4. For **Command**, choose Update Custom Cookbooks. This command installs the updated cookbook on the instance.
5. Choose **Update Custom Cookbooks**. The command might take a few minutes to finish.

Add the Recipe to the Custom IIS Layer

As with `install.rb`, the preferred way to handle deployment is to assign `deploy.rb` to the appropriate lifecycle event. You usually assign deployment recipes to the Deploy event, and they are referred to collectively as Deploy recipes. Assigning a recipe to the Deploy event does not trigger the event. Instead:

- For new instances, OpsWorks Stacks automatically runs the Deploy recipes after the Setup recipes have finished, so new instances automatically have the current application version.
- For online instances, you use a deploy command to manually install new or updated applications.

  This command triggers a Deploy event on the stack's instances, which runs the Deploy recipes.

To assign deploy.rb to the layer's Deploy event

1. Choose **Layers** in the navigation pane, and then choose **Recipes** under **Layer IISExample**.
2. Under **Custom Chef Recipes**, add **`iis-cookbook::deploy`** to the **Deploy** recipes box and choose **+** to add the recipe to the layer.
3. Choose **Save** to save the new configuration. The custom Deploy recipes should now include `iis-cookbook::deploy`.

Add an App

The final task is to add an app to the stack to represent your application in the OpsWorks Stacks environment. An app includes metadata such as the application's display name, and the data that is required to download the app from its repository.

To add the app to the stack

1. Choose **Apps** in the navigation pane, and then choose **Add an app**.
2. Configure the app with the following settings.

- **Name** – **IIS-Example-App**
- **Repository Type** – **Other**
- **Environment Variables** – Add the following three environment variables:
  - **S3REGION** – The bucket's region (in this case, `us-west-1`).
  - **BUCKET** – The bucket name, such as `windows-example-app`.
  - **FILENAME** – The file name: **default.htm**.

3. Accept default values for the remaining settings, and then choose **Add App** to add the app to the stack.

> **Note**
>
> This example uses environment variables to provide the download data. An alternative approach is to use an S3 Archive repository type and provide the file's URL. OpsWorks Stacks adds the information, along with optional data, such as your AWS credentials, to the app's `app_source` attribute. Your deploy recipe must get the URL from the app attributes and parse it to extract the region, bucket name, and file name.

Deploy the App and Open the Application

OpsWorks Stacks automatically deploys apps to new instances, but not to online instances. Because your instance is already running, you must deploy the app manually.

To deploy the app

1. Choose **Apps** in the navigation pane, and then choose **deploy** in the app's **Actions** column.
2. **Command** should be set to **Deploy**. Choose **Deploy** at the lower right of the **Deploy App** page. The command might take a few minutes to finish.

   After deployment is finished, you return to the **Apps** page. The **Status** indicator shows **successful** in green, and the app name has a green check mark next to it to indicate a successful deployment.

Note

Windows apps are always the **Other** app type, so deploying the app does the following:

- Adds the app's data to the stack configuration and deployment attributes, as described earlier.
- Triggers a Deploy event on the stack's instances, which runs your custom Deploy recipes.

Note

For more information about how to troubleshoot failed deployments or applications, see Debugging Recipes.

The app is now installed. You can open it by choosing **Instances** in the **Navigation** pane, and then choosing the instance's public IP address. This sends an HTTP request to the instance, and you should see something like the following in your browser.



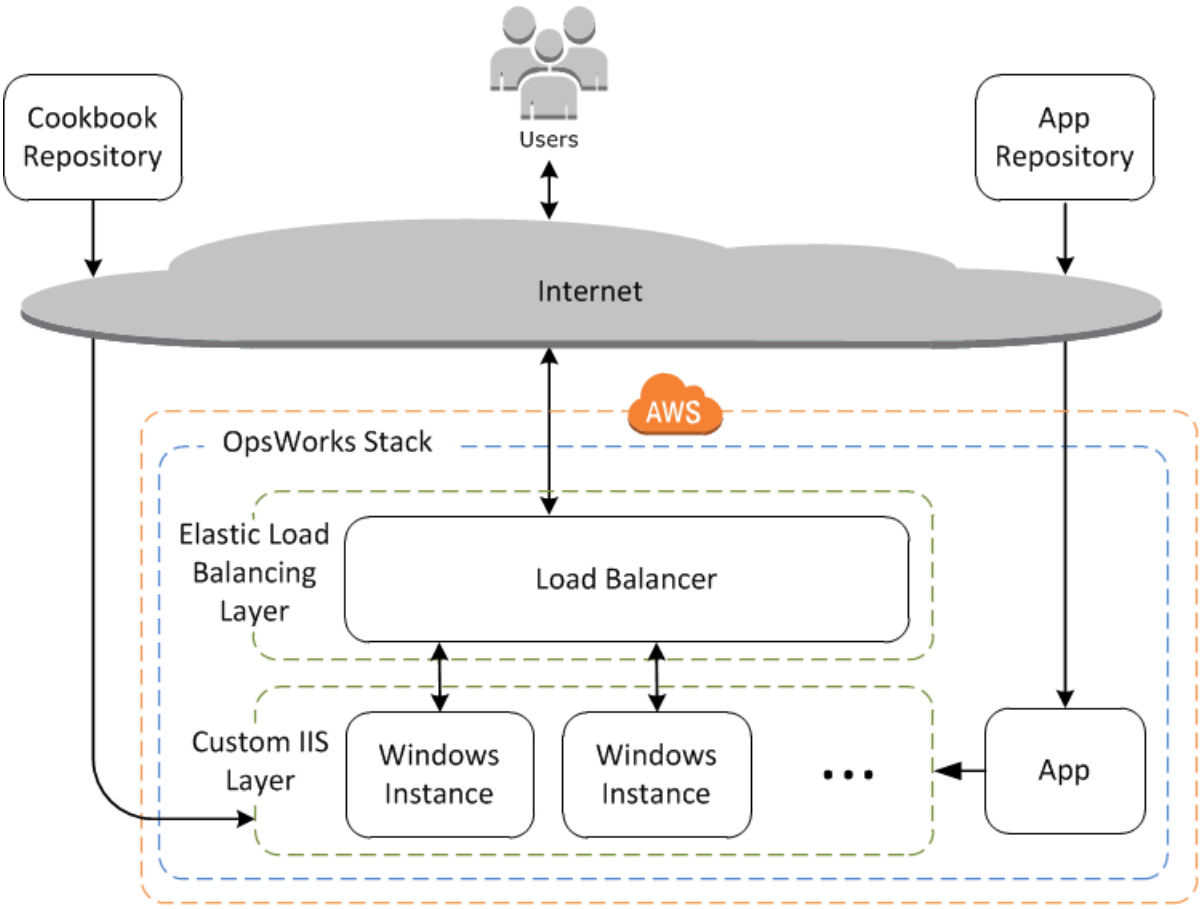
Step 3: Scale Out IISExample

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

If your incoming user requests start to approach the limit of what you can handle with a single t2.micro instance, you will need to increase your server capacity. You can move to a larger instance,

but that has limits. A more flexible approach is to add instances to your stack, and put them behind a load balancer. The basic architecture looks something like the following.



Among other advantages, this approach is much more robust than a single large instance.

- If one of your instances fails, the load balancer will distribute incoming requests to the remaining instances, and your application will continue to function.
- If you put instances in different Availability Zones (the recommended practice), your application will continue to function even if an Availability Zone encounters problems.

OpsWorks Stacks makes it easy to scale out stacks. This section describes the basics of how to scale out a stack by adding a second 24/7 PHP App Server instance to IISExample and putting both instances behind an Elastic Load Balancing load balancer. You can easily extend the procedure to add an arbitrary number of 24/7 instances, or you can use time-based instances to have OpsWorks Stacks scale your stack automatically. For more information, see Managing load with time-based and load-based instances.

Add a Load Balancer

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Elastic Load Balancing is an AWS service that automatically distributes incoming application traffic across multiple Amazon EC2 instances. A load balancer can serve two purposes. The obvious one is to equalize the load on your application servers. Many sites prefer to isolate their application servers and databases from direct user access. In addition to distributing traffic, Elastic Load Balancing does the following:

- Detects unhealthy Amazon EC2 instances.

  It reroutes traffic to the remaining healthy instances until the unhealthy instances have been restored.
- Automatically scales request handling capacity in response to incoming traffic.

> **Note**
>
> OpsWorks Stacks does not support Application Load Balancer. You can only use Classic Load Balancer with OpsWorks Stacks.

Although Elastic Load Balancing is often referred to as a layer, it works a bit differently than the other built-in layers. Instead of creating a layer and adding instances to it, you create an Elastic Load Balancing load balancer by using the Amazon EC2 console and then attach it to one of your existing layers, usually an application server layer. OpsWorks Stacks then registers the layer's existing instances with the service and automatically adds any new instances. The following procedure describes how to add a load balancer.

To attach a load balancer to the custom IIS layer

1. Use the Amazon EC2 console to create a new load balancer for IISExample. For more information, see Getting Started with Elastic Load Balancing. When you run the **Create Load Balancer** wizard, configure the load balancer as follows:

   1: Define Load Balancer

   Assign the load balancer an easily recognizable name, like IIS-LB, to make it easier to locate in the OpsWorks Stacks console. Accept the defaults for the remaining settings, and then choose **Next: Assign Security Groups**.

   2: Assign Security Groups

   If your account supports default VPC, the wizard displays this page to determine the load balancer's security group. It does not display this page for EC2 Classic.

   For this walkthrough, specify **default VPC security group**, and then choose **Next: Configure Security Settings**.

   3: Configure Security Settings

   This walkthrough does require your load balancer to use a secure listener (that is, HTTPS or SSL on its front-end connection), so choose **Next: Configure Health Check** to continue.

   4: Configure Health Check

   Set the ping path to **/**. Accept the defaults for the remaining settings, and then choose **Next: Add EC2 Instances**.

   5: Add EC2 Instances

   OpsWorks Stacks automatically takes care of registering instances with the load balancer. Choose **Next Add Tags** to continue.

   6: Add Tags

   You won't be using tags for this example. Choose **Review and Create**.

   7: Review

   Review your choices and choose **Create** and then **Close**, which launches the load balancer.

2. If your account supports default VPC, after you launch the load balancer you must ensure that its security group has appropriate inbound rules. The default rule does not accept any inbound traffic.

    1. Choose **Security Groups** in the Amazon EC2 navigation pane.
    2. Choose **default VPC security group**
    3. On the **Inbound** tab, choose **Edit** .
    4. For this walkthrough, set **Source** to **Anywhere**, which directs the load balancer to accept incoming traffic from any IP address.
    5. Click **Save**.
3. Return to the OpsWorks Stacks console. On the **Layers** page, choose **Network**.
4. Under **Elastic Load Balancing**, select the IIS-LB load balancer that you created in Step 1, and then click **Save**.

    After you have attached the load balancer to the layer, OpsWorks Stacks automatically registers the layer's current instances and adds new instances as they come online.
5. On the **Layers** page, click the load balancer's name to open its details page. A green check next to the instance on the load balancer page indicates that the instance has passed a health check.

You can now run IIS-Example-App by sending a request to the load balancer.

To run IIS-Example-App through the load balancer

1. Choose **Layers**. The IIS-ELB load balancer should be listed as a layer and the Health column should have one instance in green, which indicates a healthy instance.
2. Choose the load balancer's DNS name to run IIS-Example-App. It should be listed under the load balancer's name and look something like `IIS-LB-1802910859.us-west-2.elb.amazonaws.com`. The load balancer forwards the request to the instance and returns the response, which should look exactly the same as the response you get when you click the instance's public IP address.

You have only one instance at this point, so the load balancer isn't really adding much. However, you can now add additional instances to the layer.

To add an instance to the layer

1. Choose **Instances** and then **+ instance** to add another instance to the layer.
2. Start the instance.

Because they are new instances, OpsWorks Stacks automatically installs the current custom cookbooks and deploys the current app version during setup. When the instance comes online, OpsWorks Stacks automatically adds it to the load balancer, so your instance will immediately start handling requests. To verify that the application is still working, you can choose the load balancer's DNS name again .

Next Steps

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This walkthrough took you through the basics of setting up a simple Windows application server stack. Here are some suggestions for what to do next.

- If you would like to know more, Getting Started: Cookbooks provides a tutorial introduction to implementing cookbooks, and includes a number of OpsWorks Stacks-specific examples.
- You can add an Amazon Relational Database Service (Amazon RDS) layer to the stack to use as a backend database server. For information about how to connect your application to the database, see Using a Custom Recipe.

Getting Started with Cookbooks in OpsWorks Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate

their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

A production-level OpsWorks Stacks stack typically requires some customization, which often means implementing a custom Chef cookbook. A *cookbook* is a package file that contains configuration information, including instructions called *recipes*. A *recipe* is a set of one or more instructions, written with Ruby language syntax, that specifies the resources to use and the order in which those resources are applied. A *resource*, as used in Chef, is a statement of configuration policy. This walkthrough provides a basic introduction to implementing Chef cookbooks for OpsWorks Stacks. To learn more about Chef, cookbooks, recipes, and resources, see the links in Next Steps.

This walkthrough mostly describes how to create your own cookbooks. You can also use community-provided cookbooks available on websites like the Chef Supermarket. To help you get started with community cookbooks, we include instructions for using a community cookbook from the Chef Supermarket later in the walkthrough.

Before you start this walkthrough, complete a few setup steps. If you have already completed any of the other walkthroughs in this chapter, such as Getting Started: Sample, then you have met the prerequisites for this walkthrough and can skip to start this walkthrough. Otherwise, be sure to complete the prerequisites, and then return to this walkthrough.

Topics

- Step 1: Create the Cookbook
- Step 2: Create the Stack and its Components
- Step 3: Run and Test the Recipe
- Step 4: Update the Cookbook to Install a Package
- Step 5: Update the Cookbook on the Instance and Run the Recipe
- Step 6: Update the Cookbook to Add a User
- Step 7: Update the Cookbook to Create a Directory
- Step 8: Update the Cookbook to Create and Copy Files
- Step 9: Update the Cookbook to Run a Command
- Step 10: Update the Cookbook to Run a Script

- Step 11: Update the Cookbook to Manage a Service
- Step 12: Update the Cookbook to Use Custom JSON
- Step 13: Update the Cookbook to Use Data Bags
- Step 14: Update the Cookbook to Use Iteration
- Step 15: Update the Cookbook to Use Conditional Logic
- Step 16: Update the Cookbook to Use Community Cookbooks
- Step 17: (Optional) Clean Up
- Next Steps

Step 1: Create the Cookbook

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Start by creating a cookbook. This cookbook won't do much to start, but it serves as a foundation for the rest of this walkthrough.

Note

This step demonstrates how to create a cookbook manually. You can create a cookbook in less time with the Chef development kit (Chef DK) by running the command **chef generate cookbook** on your local workstation. However, this command creates several folders and files that you won't need for this walkthrough.

To create the cookbook

1. On your local workstation, create a directory named `opsworks_cookbook_demo`. You can use a different name, but be sure to substitute it for `opsworks_cookbook_demo` throughout this walkthrough.

2. In the opsworks_cookbook_demo directory, create a file named metadata.rb using a text editor. Add the following code to specify the cookbook's name. For more information about metadata.rb, see metadata.rb on the Chef website.

```
name "opsworks_cookbook_demo"
```

3. In the opsworks_cookbook_demo directory, create a subdirectory named recipes. This subdirectory contains all of the recipes that you create for this walkthrough's cookbook.

4. In the recipes directory, create a file named default.rb. This file contains a recipe with the same name as the file, but without the file extension: default. Add the following single line of code to the default.rb file. This code is a one-line recipe that displays a simple message in the log when the recipe runs:

```
Chef::Log.info("********** Hello, World! **********")
```

5. At the terminal or command prompt, use the **tar** command to create a file named opsworks_cookbook_demo.tar.gz, which contains the opsworks_cookbook_demo directory and its contents. For example:

```
tar -czvf opsworks_cookbook_demo.tar.gz opsworks_cookbook_demo/
```

   You can use a different file name, but be sure to substitute it for opsworks_cookbook_demo.tar.gz throughout this walkthrough.

   > **Note**
   >
   > When you create the tar file on Windows, the top directory must be the parent directory of the cookbook. This walkthrough has been tested on Linux with the **tar** command provided by the tar package and on Windows with the **tar** command provided by Git Bash. Using other commands or programs to create a compressed TAR (.tar.gz) file may not work as expected.

6. Create an S3 bucket, or use an existing bucket. For more information, see Create a Bucket.

7. Upload the opsworks_cookbook_demo.tar.gz file to the S3 bucket. For more information, see Add an Object to a Bucket.

You now have a cookbook that you will use throughout this walkthrough.

In the next step, you create an OpsWorks Stacks stack that you will use later to upload your cookbook and to run the cookbook's recipes.

Step 2: Create the Stack and its Components

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Create an OpsWorks Stacks stack and its components, which include a layer and an instance. In later steps, you upload your cookbook to the instance and then run the cookbook's recipes on that instance.

To create the stack

1. Sign in to the OpsWorks Stacks console at https://console.aws.amazon.com/opsworks.
2. Do one of the following, if they apply:
   - If the **Welcome to OpsWorks Stacks** page is displayed, choose **Add your first stack** or **Add your first OpsWorks Stacks stack** (both choices do the same thing). The **Add stack** page is displayed.
   - If the **OpsWorks Dashboard** page is displayed, choose **Add stack**. The **Add Stack** page is displayed.
3. Choose **Chef 12 stack**.
4. In the **Stack name** box, type the stack's name, for example **`MyCookbooksDemoStack`**. You can type a different name, but be sure to substitute it for `MyCookbooksDemoStack` throughout this walkthrough.
5. For **Region**, choose **US West (Oregon)**.
6. For **VPC**, do one of the following:
   - If a VPC is available, choose it. For more information, see Running a Stack in a VPC.
   - Otherwise, choose **No VPC**.

7. For **Use custom Chef cookbooks**, choose **Yes**.
8. For **Repository type**, choose **S3 Archive**.

   > **Note**
   >
   > In the Getting Started: Linux walkthrough, you chose **Http Archive**. Be sure to choose **S3 Archive** here instead.

9. For **Repository URL**, type the path to your `opsworks_cookbook_demo.tar.gz` file in S3. To get the path, in the S3 console, select the `opsworks_cookbook_demo.tar.gz` file. On the **Properties** pane, copy the value of the **Link** field. (It should be similar to this: `https://s3.amazonaws.com/amzn-s3-demo-bucket/opsworks_cookbook_demo.tar.gz`.)
10. If your S3 bucket is private, which is the default, then for **Access key ID** and **Secret access key**, type the access key ID and secret access key of the IAM user that you are using for this walkthrough. For more information, see Editing Object Permissions and Share an Object with Others.
11. Leave the defaults for the following:

    - **Default Availability Zone** (**us-west-2a**)
    - **Default operating system** (**Linux** and **Amazon Linux 2016.09**)
    - **Default SSH key** (**Do not use a default SSH key**)
    - **Stack color** (dark blue)

12. Choose **Advanced**.
13. For **IAM role**, do one of the following:

    - If **aws-opsworks-service-role** is available, choose it.
    - If **aws-opsworks-service-role** is not available, choose **New IAM role**.

14. For **Default IAM instance profile**, do one of the following:

    - If **aws-opsworks-ec2-role** is available, choose it.
    - If **aws-opsworks-ec2-role** is not available, choose **New IAM instance profile**.

15. Leave the defaults for the following:

    - **Default root device type** (**EBS backed**)
    - **Hostname theme** (**Layer Dependent**)

- **OpsWorks Agent version** (most recent version)
- **Custom Chef JSON** (blank)
- **Security**, **Use OpsWorks security groups** (**Yes**)

16. Choose **Add stack**. OpsWorks Stacks creates the stack and displays the **MyCookbooksDemoStack** page.

To create the layer

1. In the service navigation pane, choose **Layers**. The **Layers** page is displayed.
2. Choose **Add a layer**.
3. On the **OpsWorks** tab, for **Name**, type **`MyCookbooksDemoLayer`**. You can type a different name, but be sure to substitute it for `MyCookbooksDemoLayer` throughout this walkthrough.
4. For **Short name**, type **`cookbooks-demo`**. You can type a different name, but be sure to substitute it for `cookbooks-demo` throughout this walkthrough.
5. Choose **Add layer**. OpsWorks Stacks adds the layer and displays the **Layers** page.

To create and start the instance

1. In the service navigation pane, choose **Instances**. The **Instances** page is displayed.
2. Choose **Add an instance**.
3. On the **New** tab, choose **Advanced**.
4. Leave the defaults for the following:

   - **Hostname** (**cookbooks-demo1**)
   - **Size** (**c3.large**)
   - **Subnet** (*`IP address`* **us-west-2a**)
   - **Scaling type** (**24/7**)
   - **SSH key** (**Do not use a default SSH key**)
   - **Operating system** (**Amazon Linux 2016.09**)
   - **OpsWorks Agent version** (**Inherit from stack**)
   - **Tenancy** (**Default - Rely on VPC settings**)
   - **Root device type** (**EBS backed**)
   - **Volume type** (**General Purpose (SSD)**)

- **Volume size** (**8**)

5. Choose **Add instance**.
6. For **MyCookbooksDemoLayer**, for **cookbooks-demo1**, for **Actions**, choose **start**. Do not proceed until **Status** changes to **online**. This process might take several minutes, so be patient.

You now have a stack, a layer, and an instance to which the cookbook was automatically copied from your S3 bucket. In the next step, you will run and test the default recipe from within the cookbook on the instance.

Step 3: Run and Test the Recipe

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Run and test the `default` recipe from within the cookbook that OpsWorks Stacks copied to the instance. As you'll recall, this is the one-line recipe that displays a simple message in the log when the recipe runs.

To run the recipe

1. In the service navigation pane, choose **Stack**. The **MyCookbooksDemoStack** page is displayed.
2. Choose **Run Command**. The **Run Command** page is displayed.
3. For **Command**, choose **Execute Recipes**.
4. For **Recipes to execute**, type **`opsworks_cookbook_demo::default`**.

   `opsworks_cookbook_demo` is the name of the cookbook as defined in the `metadata.rb` file. **`default`** is the name of the recipe to run, that is, the name of the `default.rb` file in the cookbook's `recipes` subdirectory, without the file extension.
5. Leave the following default settings:

   - **Comment** (blank)

- **Advanced**, **Custom Chef JSON** (blank)
- **Instances** (**Select all** checked, **MyCookbooksDemoLayer** checked, **cookbooks-demo1** checked)

6. Choose **Execute Recipes**. The **Running command execute_recipes** page is displayed. Do not proceed until **Status** changes to **successful**. This process might take a few minutes, so be patient.

To check the recipe's results

1. With the **Running command execute_recipes** page displayed, for **cookbooks-demo1**, for **Log**, choose **show**. The **execute_recipes** log page is displayed.
2. Scroll down the log and find an entry that looks similar to the following:

```
[2015-11-13T19:14:39+00:00] INFO: ********** Hello, World! **********
```

You have successfully run your first recipe! In the next step, you will update your cookbook by adding a recipe that installs a package on the instance.

Step 4: Update the Cookbook to Install a Package

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Update your cookbook by adding a recipe that installs on the instance a package that contains the popular text editor GNU Emacs.

Although you can just as easily log in to the instance and install the package once, writing a recipe enables you to run the recipe from OpsWorks Stacks once to install multiple packages on multiple instances in a stack simultaneously.

To update the cookbook to install a package

1. Back on your local workstation, in the `recipes` subdirectory in the `opsworks_cookbook_demo` directory, create a file named `install_package.rb` with the following code:

   ```
   package "Install Emacs" do
     package_name "emacs"
   end
   ```

   This recipe installs the `emacs` package on the instance. (For more information, go to package.)

   > **Note**
   >
   > You can give a recipe any file name you want. Just be sure to specify the correct recipe name whenever you want OpsWorks Stacks to run the recipe.

2. At the terminal or command prompt, use the **tar** command create a new version of the `opsworks_cookbook_demo.tar.gz` file, which contains the `opsworks_cookbook_demo` directory and its updated contents.
3. Upload the updated `opsworks_cookbook_demo.tar.gz` file to your S3 bucket.

This new recipe runs when you update the cookbook on the instance and then run the new recipe from within the updated cookbook. The next step describes how to do this.

After you complete the next step, you will be able to log in to the instance and then type **emacs** from the command prompt to launch GNU Emacs. (For more information, see Connect to Your Linux Instance.) To exit GNU Emacs, press **Ctrl+X**, then **Ctrl+C**.

> **Important**
>
> To log in to the instance, you must first provide OpsWorks Stacks with information about your public SSH key (which you can create with tools such as ssh-keygen or PuTTYgen), and then you must set permissions on the `MyCookbooksDemoStack` stack to enable your user to log in to the instance. For instructions, see Registering a User's Public SSH Key and Logging In with SSH.

Step 5: Update the Cookbook on the Instance and Run the Recipe

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Update the cookbook on the instance and then run the recipe from within the updated cookbook on the instance. Throughout the rest of this walkthrough, you repeat this step every time you update the cookbook by adding a new recipe.

To update the cookbook on the instance

1. In the service navigation pane, choose **Stack**. The **MyCookbooksDemoStack** page is displayed.
2. Choose **Run Command**. The **Run Command** page is displayed.
3. For **Command**, choose **Update Custom Cookbooks**.
4. Leave the following default settings:
   - **Comment** (blank)
   - **Advanced**, **Custom Chef JSON** (blank)
   - **Advanced**, **Instances** (**Select all** checked, **MyCookbooksDemoLayer** checked, **cookbooks-demo1** checked)
5. Choose **Update Custom Cookbooks**. The **Running command update_custom_cookbooks** page is displayed. Do not proceed until **Status** changes to **successful**. This process might take several minutes, so be patient.

To run the recipe

1. In the service navigation pane, choose **Stack**. The **MyCookbooksDemoStack** page is displayed.
2. Choose **Run Command**. The **Run Command** page is displayed.
3. For **Command**, choose **Execute Recipes**.

4. For **Recipes to execute**, type the name of the recipe to run. The first time you do this, the recipe is named **`opsworks_cookbook_demo::install_package`**.

   > **Note**
   >
   > As you repeat this procedure later, type the name of the cookbook (**`opsworks_cookbook_demo`**), followed by two colons (**`::`**), followed by the name of the recipe (the recipe's file name, without the `.rb` file extension).

5. Leave the following default settings:

   - **Comment** (blank)
   - **Advanced**, **Custom Chef JSON** (blank)
   - **Instances Select all** checked, **MyCookbooksDemoLayer** checked, **cookbooks-demo1** checked)

6. Choose **Execute Recipes**. The **Running command execute_recipes** page is displayed. Do not proceed until **Status** changes to **successful**. This process might take a few minutes, so be patient.

> **Note**
>
> You don't have to manually run recipes. You can assign recipes to a layer's lifecycle events, such as the Setup and Configure events, and OpsWorks Stacks will run those recipes automatically when the event occurs. For more information, see OpsWorks Stacks Lifecycle Events.

In the next step, you will update the cookbook to add a user to the instance.

Step 6: Update the Cookbook to Add a User

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Update your cookbook by adding a recipe that adds a local user to the instance and sets the user's home directory and shell. This is similar to running the Linux **adduser** or **useradd** commands or the Windows **net user** command. You add a local user to an instance, for example, when you want to control access to the instance's files and directories.

You can also manage users without using cookbooks. For more information, see Managing Users.

To update the cookbook on the instance and to run the new recipe

1. On your local workstation, in the `recipes` subdirectory in the `opsworks_cookbook_demo` directory, create a file named `add_user.rb` with the following code (for more information, go to user):

   ```
   user "Add a user" do
     home "/home/jdoe"
     shell "/bin/bash"
     username "jdoe"
   end
   ```

2. At the terminal or command prompt, use the **tar** command create a new version of the `opsworks_cookbook_demo.tar.gz` file, which contains the `opsworks_cookbook_demo` directory and its updated contents.
3. Upload the updated `opsworks_cookbook_demo.tar.gz` file to your S3 bucket.
4. Follow the procedures in Step 5: Update the Cookbook on the Instance and Run the Recipe to update the cookbook on the instance and to run the recipe. In the "To run the recipe" procedure, for **Recipes to execute**, type **`opsworks_cookbook_demo::add_user`**.

To test the recipe

1. Log in to the instance, if you have not done so already.
2. From the command prompt, run the following command to confirm that the new user was added:

   ```
   grep jdoe /etc/passwd
   ```

Information similar to the following is displayed about the user, including details such as the user's name, ID number, group ID number, home directory, and shell:

```
jdoe:x:501:502::/home/jdoe:/bin/bash
```

In the next step, you will update the cookbook to create a directory on the instance.

Step 7: Update the Cookbook to Create a Directory

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Update your cookbook by adding a recipe that adds a directory to the instance. This is similar to running the Linux **mkdir** command or the Windows **md** or **mkdir** commands.

To update the cookbook on the instance and to run the new recipe

1. On your local workstation, in the `recipes` subdirectory in the `opsworks_cookbook_demo` directory, create a file named `create_directory.rb` with the following code. For more information, go to directory:

   ```
   directory "Create a directory" do
     group "root"
     mode "0755"
     owner "ec2-user"
     path "/tmp/create-directory-demo"
   end
   ```

2. At the terminal or command prompt, use the **tar** command create a new version of the `opsworks_cookbook_demo.tar.gz` file, which contains the `opsworks_cookbook_demo` directory and its updated contents.
3. Upload the updated `opsworks_cookbook_demo.tar.gz` file to your S3 bucket.

4. Follow the procedures in Step 5: Update the Cookbook on the Instance and Run the Recipe to update the cookbook on the instance and to run the recipe. In the "To run the recipe" procedure, for **Recipes to execute**, type **`opsworks_cookbook_demo::create_directory`**.

To test the recipe

1. Log in to the instance, if you have not done so already.
2. From the command prompt, run the following command to confirm that the new directory was added:

   ```
   ls -la /tmp/create-directory-demo
   ```

   Information about the newly-added directory is displayed, including information such as permissions, owner name, and group name:

   ```
   drwxr-xr-x 2 ec2-user root 4096 Nov 18 00:35 .
   drwxrwxrwt 6 root     root 4096 Nov 24 18:17 ..
   ```

In the next step, you will update the cookbook to create a file on the instance.

Step 8: Update the Cookbook to Create and Copy Files

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Update your cookbook by adding a recipe that adds two files to the instance. The first resource in the recipe creates a file completely with recipe code. This is similar to running the Linux **cat**, **echo**, or **touch** commands or the Windows **echo** or **fsutil** commands. This technique is useful for a few, small, or simple files. The second resource in the recipe copies a file in the cookbook to another directory on the instance. This is similar to running the Linux **cp** command or the Windows **copy** command.This technique is useful for many, large, or complex files.

Before you start this step, complete Step 7: Update the Cookbook to Create a Directory to make sure that the files' parent directory already exists.

To update the cookbook on the instance and run the new recipe

1. On your local workstation, in the `opsworks_cookbook_demo` directory, create a subdirectory named `files`.
2. In the `files` subdirectory, create a file named `hello.txt` with the following text: **`Hello, World!`**
3. In the `recipes` subdirectory in the `opsworks_cookbook_demo` directory, create a file named `create_files.rb` with the following code. For more information, go to file and cookbook_file.

   ```
   file "Create a file" do
     content "<html>This is a placeholder for the home page.</html>"
     group "root"
     mode "0755"
     owner "ec2-user"
     path "/tmp/create-directory-demo/index.html"
   end

   cookbook_file "Copy a file" do
     group "root"
     mode "0755"
     owner "ec2-user"
     path "/tmp/create-directory-demo/hello.txt"
     source "hello.txt"
   end
   ```

   The `file` resource creates a file in the specified path. The `cookbook_file` resource copies the file from the `files` directory that you just created in the cookbook (Chef expects to find a standard-named subdirectory named `files` that it can copy files from) to another directory on the instance.
4. At the terminal or command prompt, use the **tar** command create a new version of the `opsworks_cookbook_demo.tar.gz` file, which contains the `opsworks_cookbook_demo` directory and its updated contents.
5. Upload the updated `opsworks_cookbook_demo.tar.gz` file to your S3 bucket.

6. Follow the procedures in Step 5: Update the Cookbook on the Instance and Run the Recipe to update the cookbook on the instance and to run the recipe. In the "To run the recipe" procedure, for **Recipes to execute**, type **`opsworks_cookbook_demo::create_files`**.

To test the recipe

1. Log in to the instance, if you have not done so already.
2. From the command prompt, run the following commands, one at a time, to confirm that the new files were added:

```
sudo cat /tmp/create-directory-demo/index.html

sudo cat /tmp/create-directory-demo/hello.txt
```

   The files' contents are displayed:

```
<html>This is a placeholder for the home page.</html>

Hello, World!
```

In the next step, you will update the cookbook to run a command on the instance.

Step 9: Update the Cookbook to Run a Command

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Update your cookbook by adding a recipe that runs a command that creates an SSH key on the instance.

To update the cookbook on the instance and run the new recipe

1. On your local workstation, in the `recipes` subdirectory in the `opsworks_cookbook_demo` directory, create a file named `run_command.rb` with the following code. For more information, go to execute.

   ```
   execute "Create an SSH key" do
     command "ssh-keygen -f /tmp/my-key -N fLyC3jbY"
   end
   ```

2. At the terminal or command prompt, use the **tar** command create a new version of the `opsworks_cookbook_demo.tar.gz` file, which contains the `opsworks_cookbook_demo` directory and its updated contents.
3. Upload the updated `opsworks_cookbook_demo.tar.gz` file to your S3 bucket.
4. Follow the procedures in Step 5: Update the Cookbook on the Instance and Run the Recipe to update the cookbook on the instance and to run the recipe. In the "To run the recipe" procedure, for **Recipes to execute**, type **`opsworks_cookbook_demo::run_command`**.

To test the recipe

1. Log in to the instance, if you have not done so already.
2. From the command prompt, run the following commands, one at a time, to confirm that the SSH key was created:

   ```
   sudo cat /tmp/my-key

   sudo cat /tmp/my-key.pub
   ```

   The SSH private and public key's contents are displayed:

   ```
   -----BEGIN RSA PRIVATE KEY-----
   Proc-Type: 4,ENCRYPTED
   DEK-Info: AES-128-CBC,DEF7A09C...541583FA
   A5p9dCuo...wp0YYH1c
   -----END RSA PRIVATE KEY-----

   ssh-rsa AAAAB3N...KaNogZkT root@cookbooks-demo1
   ```

In the next step, you will update the cookbook to run a script on the instance.

Step 10: Update the Cookbook to Run a Script

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Update your cookbook by adding a recipe that runs a script on the instance. This recipe creates a directory and then creates a file in that directory. Writing a recipe to run a script that contains multiple commands is easier than running those commands one at a time.

To update the cookbook on the instance and run the new recipe

1. On your local workstation, in the `recipes` subdirectory in the `opsworks_cookbook_demo` directory, create a file named `run_script.rb` with the following code. For more information, go to script.

   ```
   script "Run a script" do
     interpreter "bash"
     code <<-EOH
       mkdir -m 777 /tmp/run-script-demo
       touch /tmp/run-script-demo/helloworld.txt
       echo "Hello, World!" > /tmp/run-script-demo/helloworld.txt
     EOH
   end
   ```

2. At the terminal or command prompt, use the **tar** command create a new version of the `opsworks_cookbook_demo.tar.gz` file, which contains the `opsworks_cookbook_demo` directory and its updated contents.
3. Upload the updated `opsworks_cookbook_demo.tar.gz` file to your S3 bucket.
4. Follow the procedures in Step 5: Update the Cookbook on the Instance and Run the Recipe to update the cookbook on the instance and to run the recipe. In the "To run the recipe" procedure, for **Recipes to execute**, type **`opsworks_cookbook_demo::run_script`**.

To test the recipe

1. Log in to the instance, if you have not done so already.
2. From the command prompt, run the following command to confirm that the new file was added:

   ```
   sudo cat /tmp/run-script-demo/helloworld.txt
   ```

   The file's contents are displayed:

   ```
   Hello, World!
   ```

In the next step, you will update the cookbook to manage a service on the instance.

Step 11: Update the Cookbook to Manage a Service

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Update your cookbook by adding a recipe that manages a service on the instance. This is similar to running the Linux **service** command or the Windows **net stop**, **net start**, and similar commands.This recipe stops the **crond** service on the instance.

To update the cookbook on the instance and run the new recipe

1. On your local workstation, in the `recipes` subdirectory in the `opsworks_cookbook_demo` directory, create a file named `manage_service.rb` with the following code. For more information, go to service.

   ```
   service "Manage a service" do
     action :stop
     service_name "crond"
   ```

```
end
```

2. At the terminal or command prompt, use the **tar** command create a new version of the `opsworks_cookbook_demo.tar.gz` file, which contains the `opsworks_cookbook_demo` directory and its updated contents.
3. Upload the updated `opsworks_cookbook_demo.tar.gz` file to your S3 bucket.
4. Follow the procedures in Step 5: Update the Cookbook on the Instance and Run the Recipe to update the cookbook on the instance and to run the recipe. In the "To run the recipe" procedure, for **Recipes to execute**, type **`opsworks_cookbook_demo::manage_service`**.

To test the recipe

1. Log in to the instance, if you have not done so already.
2. From the command prompt, run the following command to confirm that the **crond** service is stopped:

   ```
   service crond status
   ```

   The following is displayed:

   ```
   crond is stopped
   ```

3. To restart the **crond** service, run the following command:

   ```
   sudo service crond start
   ```

   The following is displayed:

   ```
   Starting crond:  [  OK  ]
   ```

4. To confirm that the **crond** service has started, run the following command again:

   ```
   service crond status
   ```

   Information similar to the following is displayed:

   ```
   crond (pid  3917) is running...
   ```

In the next step, you will update the cookbook to reference information stored as custom JSON on the instance.

Step 12: Update the Cookbook to Use Custom JSON

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Update your cookbook by adding a recipe that references custom JSON that is stored on the instance.

You can specify information in custom JSON format whenever you create, update, or clone a stack or when you run a deployment or stack command. This is useful, for example, for making a small, unchanging portion of data available to your recipes on the instance instead of getting this data from a database. For more information, see Using Custom JSON.

For this walkthrough, you will use custom JSON to provide some fictitious information about a customer invoice. The custom JSON is described later in this step.

To update the cookbook on the instance and run the new recipe

1. On your local workstation, in the `recipes` subdirectory in the `opsworks_cookbook_demo` directory, create a file named `custom_json.rb` that contains the following recipe code:

   ```
   Chef::Log.info("********** For customer '#{node['customer-id']}' invoice
    '#{node['invoice-number']}' **********")
   Chef::Log.info("********** Invoice line number 1 is a '#{node['line-items']
   ['line-1']}' **********")
   Chef::Log.info("********** Invoice line number 2 is a '#{node['line-items']
   ['line-2']}' **********")
   Chef::Log.info("********** Invoice line number 3 is a '#{node['line-items']
   ['line-3']}' **********")
   ```

   This recipe displays messages in the log about values in the custom JSON.

2. At the terminal or command prompt, use the **tar** command create a new version of the `opsworks_cookbook_demo.tar.gz` file, which contains the `opsworks_cookbook_demo` directory and its updated contents.
3. Upload the updated `opsworks_cookbook_demo.tar.gz` file to your S3 bucket.
4. Follow the procedures in Step 5: Update the Cookbook on the Instance and Run the Recipe to update the cookbook on the instance and to run the recipe. In the "To run the recipe" procedure, for **Recipes to execute**, type **`opsworks_cookbook_demo::custom_json`**. For **Advanced**, **Custom Chef JSON**, type the following custom JSON:

```
{
  "customer-id": "0123",
  "invoice-number": "9876",
  "line-items": {
    "line-1": "tractor",
    "line-2": "passenger car",
    "line-3": "trailer"
  }
}
```

To test the recipe

1. With the **Running command execute_recipes** page displayed from the previous procedures, for **cookbooks-demo1**, for **Log**, choose **show**. The **execute_recipes** log page is displayed.
2. Scroll down through the log to find entries that look similar to the following:

```
[2015-11-14T14:18:30+00:00] INFO: ********** For customer '0123' invoice '9876'
 **********
[2015-11-14T14:18:30+00:00] INFO: ********** Invoice line number 1 is a 'tractor'
 **********
[2015-11-14T14:18:30+00:00] INFO: ********** Invoice line number 2 is a 'passenger
 car' **********
[2015-11-14T14:18:30+00:00] INFO: ********** Invoice line number 3 is a 'trailer'
 **********
```

These entries display information from the custom JSON that was typed in the **Advanced**, **Custom Chef JSON** box.

In the next step, you will update the cookbook to get information from data bags, which are collections of stack settings that OpsWorks Stacks stores on each instance.

Step 13: Update the Cookbook to Use Data Bags

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Update your cookbook by adding a recipe that references stack settings that OpsWorks Stacks stores on the instance in a set of data bags. This recipe displays messages in the log about specific stack settings that are stored on the instance. For more information, see the OpsWorks Stacks Data Bag Reference.

To update the cookbook on the instance and to run the new recipe

1. On your local workstation, in the `recipes` subdirectory in the `opsworks_cookbook_demo` directory, create a file named `data_bags.rb` that contains the following code:

```
instance = search("aws_opsworks_instance").first
layer = search("aws_opsworks_layer").first
stack = search("aws_opsworks_stack").first

Chef::Log.info("********** This instance's instance ID is
 '#{instance['instance_id']}' **********")
Chef::Log.info("********** This instance's public IP address is
 '#{instance['public_ip']}' **********")
Chef::Log.info("********** This instance belongs to the layer '#{layer['name']}'
 **********")
Chef::Log.info("********** This instance belongs to the stack '#{stack['name']}'
 **********")
Chef::Log.info("********** This stack gets its cookbooks from
 '#{stack['custom_cookbooks_source']['url']}' **********")
```

This recipe displays messages in the log about specific stack settings that are stored on the instance.

2. At the terminal or command prompt, use the **tar** command create a new version of the `opsworks_cookbook_demo.tar.gz` file, which contains the `opsworks_cookbook_demo` directory and its updated contents.
3. Upload the updated `opsworks_cookbook_demo.tar.gz` file to your S3 bucket.
4. Follow the procedures in Step 5: Update the Cookbook on the Instance and Run the Recipe to update the cookbook on the instance and to run the recipe. In the "To run the recipe" procedure, for **Recipes to execute**, type **`opsworks_cookbook_demo::data_bags`**.

To test the recipe

1. With the **Running command execute_recipes** page displayed from the previous procedure, for **cookbooks-demo1**, for **Log**, choose **show**. The **execute_recipes** log page is displayed.
2. Scroll down through the log and find entries that look similar to the following:

```
[2015-11-14T14:39:06+00:00] INFO: ********** This instance's instance ID is
 'f80fa119-81ab-4c3c-883d-6028e52c89EX' **********
[2015-11-14T14:39:06+00:00] INFO: ********** This instance's public IP address is
 '192.0.2.0' **********
[2015-11-14T14:39:06+00:00] INFO: ********** This instance belongs to the layer
 'MyCookbooksDemoLayer' **********
[2015-11-14T14:39:06+00:00] INFO: ********** This instance belongs to the stack
 'MyCookbooksDemoStack' **********
[2015-11-14T14:39:06+00:00] INFO: ********** This stack gets its cookbooks from
 'https://s3.amazonaws.com/amzn-s3-demo-bucket/opsworks_cookbook_demo.tar.gz'
 **********
```

This recipe displays messages about specific stack settings that are stored on the instance.

In the next step, you will update the cookbook to run recipe code multiple times.

Step 14: Update the Cookbook to Use Iteration

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Update your cookbook by adding a recipe that uses *iteration*, a technique that repeats recipe code multiple times. This recipe displays messages in the log for a data bag item that contains multiple contents.

To update the cookbook on the instance and to run the new recipe

1. On your local workstation, in the `recipes` subdirectory in the `opsworks_cookbook_demo` directory, create a file named `iteration_demo.rb` that contains the following code:

```
stack = search("aws_opsworks_stack").first
Chef::Log.info("********** Content of 'custom_cookbooks_source' **********")

stack["custom_cookbooks_source"].each do |content|
  Chef::Log.info("********** '#{content}' **********")
end
```

> **Note**
>
> Writing the preceding recipe code is shorter, more flexible, and less error-prone than writing the following recipe code that does not use iteration:
>
> ```
> stack = search("aws_opsworks_stack").first
> Chef::Log.info("********** Content of 'custom_cookbooks_source' **********")
>
> Chef::Log::info("********** '[\"type\", \"#{stack['custom_cookbooks_source']
> ['type']}\"]' **********")
> Chef::Log::info("********** '[\"url\", \"#{stack['custom_cookbooks_source']
> ['url']}\"]' **********")
> ```

```
Chef::Log::info("********** '[\"username\",
 \"#{stack['custom_cookbooks_source']['username']}\"]' **********")
Chef::Log::info("********** '[\"password\",
 \"#{stack['custom_cookbooks_source']['password']}\"]' **********")
Chef::Log::info("********** '[\"ssh_key\",
 \"#{stack['custom_cookbooks_source']['ssh_key']}\"]' **********")
Chef::Log::info("********** '[\"revision\",
 \"#{stack['custom_cookbooks_source']['revision']}\"]' **********")
```

2. At the terminal or command prompt, use the **tar** command create a new version of the `opsworks_cookbook_demo.tar.gz` file, which contains the `opsworks_cookbook_demo` directory and its updated contents.
3. Upload the updated `opsworks_cookbook_demo.tar.gz` file to your S3 bucket.
4. Follow the procedures in Step 5: Update the Cookbook on the Instance and Run the Recipe to update the cookbook on the instance and to run the recipe. In the "To run the recipe" procedure, for **Recipes to execute**, type **`opsworks_cookbook_demo::iteration_demo`**.

To test the recipe

1. With the **Running command execute_recipes** page displayed from the previous procedures, for **cookbooks-demo1**, for **Log**, choose **show**. The **execute_recipes** log page is displayed.
2. Scroll down through the log and find entries that look similar to the following:

```
[2015-11-16T19:56:56+00:00] INFO: ********** Content of 'custom_cookbooks_source'
 **********
[2015-11-16T19:56:56+00:00] INFO: ********** '["type", "s3"]' **********
[2015-11-16T19:56:56+00:00] INFO: ********** '["url", "https://s3.amazonaws.com/
amzn-s3-demo-bucket/opsworks_cookbook_demo.tar.gz"]' **********
[2015-11-16T19:56:56+00:00] INFO: ********** '["username", "secret-key-value"]'
 **********
[2015-11-16T19:56:56+00:00] INFO: ********** '["password", "secret-access-key-
value"]' **********
[2015-11-16T19:56:56+00:00] INFO: ********** '["ssh_key", nil]' **********
[2015-11-16T19:56:56+00:00] INFO: ********** '["revision", nil]' **********
```

This recipe displays messages in the log for a data bag item that contains multiple contents. The data bag item is in the `aws_opsworks_stack` data bag. The data bag item has content

named `custom_cookbooks_source`. Inside of this content are six contents named `type`, `url`, `username`, `password`, `ssh_key`, and `revision`; their values are also displayed.

In the next step, you will update the cookbook to run recipe code only if certain conditions are met.

Step 15: Update the Cookbook to Use Conditional Logic

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Now update your cookbook by adding a recipe that uses *conditional logic*, a technique that runs code only if certain conditions are met. For more information, go to if Statements and case Statements.

This recipe does two things based on data bag content: displays a message in the log identifying the operating system that the instance is running on and, only if the operating system is Linux, installs a package by using the correct package manager for the given Linux distribution. This package is named tree; it is a simple app for visualizing directory lists.

To update the cookbook on the instance and to run the new recipe

1. On your local workstation, in the `recipes` subdirectory in the `opsworks_cookbook_demo directory`, create a file named `conditional_logic.rb` that contains the following code:

```
instance = search("aws_opsworks_instance").first
os = instance["os"]

if os == "Red Hat Enterprise Linux 7"
  Chef::Log.info("********** Operating system is Red Hat Enterprise Linux.
 **********")
elsif os == "Ubuntu 14.04 LTS" || os == "Ubuntu 16.04 LTS" || os == "Ubuntu 18.04
 LTS"
  Chef::Log.info("********** Operating system is Ubuntu. **********")
elsif os == "Microsoft Windows Server 2012 R2 Base"
```

```
  Chef::Log.info("********** Operating system is Windows. **********")
elsif os == "Amazon Linux 2015.03" || os == "Amazon Linux 2015.09" || os == "Amazon
 Linux 2016.03" || os == "Amazon Linux 2016.09" || os == "Amazon Linux 2017.03"
 || os == "Amazon Linux 2017.09" || os == "Amazon Linux 2018.03" || os == "Amazon
 Linux 2"
  Chef::Log.info("********** Operating system is Amazon Linux. **********")
elsif os == "CentOS Linux 7"
  Chef::Log.info("********** Operating system is CentOS 7. **********")
else
  Chef::Log.info("********** Cannot determine operating system. **********")
end

case os
when "Ubuntu 14.04 LTS", "Ubuntu 16.04 LTS", "Ubuntu 18.04 LTS"
  apt_package "Install a package with apt-get" do
    package_name "tree"
  end
when "Amazon Linux 2015.03", "Amazon Linux 2015.09", "Amazon Linux 2016.03",
 "Amazon Linux 2016.09", "Amazon Linux 2017.03", "Amazon Linux 2017.09", "Amazon
 Linux 2018.03", "Amazon Linux 2", "Red Hat Enterprise Linux 7", "CentOS Linux 7"
  yum_package "Install a package with yum" do
    package_name "tree"
  end
else
  Chef::Log.info("********** Cannot determine operating system type, or operating
 system is not Linux. Package not installed. **********")
end
```

2. At the terminal or command prompt, use the **tar** command create a new version of the `opsworks_cookbook_demo.tar.gz` file, which contains the `opsworks_cookbook_demo` directory and its updated contents.
3. Upload the updated `opsworks_cookbook_demo.tar.gz` file to your S3 bucket.
4. Follow the procedures in Step 5: Update the Cookbook on the Instance and Run the Recipe to update the cookbook on the instance and to run the recipe. In the "To run the recipe" procedure, for **Recipes to execute**, type **`opsworks_cookbook_demo::conditional_logic`**.

To test the recipe

1. With the **Running command execute_recipes** page displayed from the previous procedures, for **cookbooks-demo1**, for **Log**, choose **show**. The **execute_recipes** log page is displayed.
2. Scroll down through the log and find an entry that looks similar to the following:

```
[2015-11-16T19:59:05+00:00] INFO: ********** Operating system is Amazon Linux.
 **********
```

Because the instance's operating system is Amazon Linux 2016.09, only the preceding entry (of the five possible entries in the recipe's code) will be displayed in the log.

3. If the operating system is Linux, the recipe installs the tree package. To see a visualization of a directory's contents, type **`tree`** at the command prompt from the desired directory or with the desired directory's path (for example, `tree /var/chef/runs`).

In the next step, you will update the cookbook to use functionality from an external cookbook provided by the Chef community.

Step 16: Update the Cookbook to Use Community Cookbooks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Finally, update the cookbook to use functionality provided in an external cookbook provided by the Chef community. The external cookbook that you will use for this walkthrough is available through the Chef Supermarket, a popular location for accessing external Chef cookbooks. This external cookbook provides a custom resource that lets you download and install applications, similar to what you did in Step 4: Update the Cookbook to Install a Package. However, this resource can install web applications and other application types in addition to packages.

When a cookbook depends on another cookbook, you must specify a dependency on the other cookbook. To declare and manage cookbook dependencies, we recommend that you use a tool called Berkshelf. For more information about how to install Berkshelf on your local workstation, see About Berkshelf on the Chef website.

After you install Berkshelf, follow these procedures to declare the cookbook dependency and then create a recipe that calls the resource in the external cookbook:

To declare the cookbook dependency

1. On your local workstation, in the `opsworks_cookbook_demo` directory, add the following line at the end of the `metadata.rb` file:

   ```
   depends "application", "5.0.0"
   ```

   This declares a dependency on a cookbook named `application`, version 5.0.0.

2. From the root of the `opsworks_cookbook_demo` directory, run the following command. The period at the end of the command is intentional.

   ```
   berks init .
   ```

   Berkshelf creates a number of folders and files that you can use later for more advanced scenarios. The only file that we need for this walkthrough is the file named `Berksfile`.

3. Add the following line at the end of the `Berksfile` file:

   ```
   cookbook "application", "5.0.0"
   ```

   This informs Berkshelf that you want to use application cookbook version 5.0.0, which Berkshelf downloads from the Chef Supermarket.

4. At the terminal or command prompt, run the following command from the root of the `opsworks_cookbook_demo` directory:

   ```
   berks install
   ```

   Berkshelf creates a list of the dependencies for both your cookbook and the application cookbook. Berkshelf uses this list of dependencies in the next procedure.

To update the cookbook on the instance and to run the new recipe

1. In the `recipes` subdirectory in the `opsworks_cookbook_demo` directory, create a file named `dependencies_demo.rb` that contains the following code:

   ```
   application "Install NetHack" do
     package "nethack.x86_64"
   ```

```
end
```

This recipe depends on the application resource from the application cookbook to install the popular text-based adventure game NetHack on the instance. (You can, of course, subtitute any other package name you want, provided the package is readily available to the package manager on the instance.)

2. From the root of the `opsworks_cookbook_demo` directory, run the following command:

```
berks package
```

Berkshelf uses the list of dependencies from the previous procedure to create a file named `cookbooks-`*`timestamp`*`.tar.gz`, which contains the `opsworks_cookbook_demo` directory and its updated contents, including the cookbook's dependent cookbooks. Rename this file `opsworks_cookbook_demo.tar.gz`.

3. Upload the updated, renamed `opsworks_cookbook_demo.tar.gz` file to your S3 bucket.
4. Follow the procedures in Step 5: Update the Cookbook on the Instance and Run the Recipe to update the cookbook on the instance and to run the recipe. In the "To run the recipe" procedure, for **Recipes to execute**, type **`opsworks_cookbook_demo::dependencies_demo`**.
5. After you run the recipe, you should be able to log in to the instance and then type **`nethack`** at the command prompt to begin playing. (For more information about the game, see NetHack and the NetHack Guidebook.)

In the next step, you can clean up the AWS resources that you used for this walkthrough. This next step is optional. You may want to keep using these AWS resources as you continue to learn more about OpsWorks Stacks. However, keeping these AWS resources around may result in some ongoing charges to your AWS account. If you want to keep these AWS resources around for later use, you have now completed this walkthrough, and you can skip ahead to Next Steps.

Step 17: (Optional) Clean Up

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

To prevent incurring additional charges to your AWS account, you can delete the AWS resources that were used for this walkthrough. These AWS resources include the S3 bucket, the OpsWorks Stacks stack, and the stack's components. (For more information, see AWS OpsWorks Pricing.) However, you might want to keep using these AWS resources as you continue to learn more about OpsWorks Stacks. If you want to keep these AWS resources available, you have now completed this walkthrough, and you can skip to Next Steps.

Content stored in the resources that you created for this walkthrough can contain personally-identifying information. If you no longer want this information to be stored by AWS, follow steps in this topic.

To delete the S3 bucket

- See Delete the Amazon S3 Bucket.

To delete the instance for the stack

1. In the OpsWorks Stacks console, in the service navigation pane, choose **Instances**. The **Instances** page is displayed.
2. For **MyCookbooksDemoLayer**, for **cookbooks-demo1**, for **Actions**, choose **stop**. When you see the confirmation message, choose **Stop**.
3. The following changes occur over several minutes. Do not proceed until all of the following have finished.
   - **Status** changes from **online** to **stopping** and eventually to **stopped**.
   - **online** changes from **1** to **0**.
   - **shutting down** changes from **0** to **1** and eventually back to **0**.
   - **stopped** eventually changes from **0** to **1**.
4. For **Actions**, choose **delete**. When you see the confirmation message, choose **Delete**. OpsWorks Stacks deletes the instance and displays **No instances**.

To delete the stack

1. In the service navigation pane, choose **Stack**. The **MyCookbooksDemoStack** page is displayed.
2. Choose **Delete Stack**. When you see the confirmation message, choose **Delete**. OpsWorks Stacks deletes the stack and displays the **Dashboard** page.

Optionally, you can delete the IAM user and Amazon EC2 key pair that you used for this walkthrough, if you don't want to reuse them for access to other AWS services and EC2 instances. For instructions, see Deleting an IAM user and Amazon EC2 key pairs and Linux instances.

You have now completed this walkthrough. For more information, see Next Steps.

Next Steps

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Now that you have completed this walkthrough, you can learn more about OpsWorks Stacks support for Chef cookbooks by reviewing the following resources:

- Cookbooks and Recipes – Describes the versions of Chef and Ruby that OpsWorks Stacks currently supports. Also demonstrates how to install and update custom cookbooks on instances and how to run recipes on instances.
- Learn Chef – Provides links to Chef tutorials, a Chef skills library, complete Chef documentation, and Chef training classes.
- All about Chef – Provides complete Chef documentation. Specific topics of interest include:
  - About Cookbooks – Describes key cookbook components such as attributes, recipes, files, metadata, and templates.
  - About Recipes – Describes the fundamentals of recipes such as how to work with data bags, include other recipes, and use Ruby code in recipes.

  - Resources – Describes how to use all of the built-in Chef resources, such as `apt_package`, `cookbook_file`, `directory`, `execute`, `file`, and `package`.
  - About the Recipe DSL – Describes how to write code for Chef recipes with statements such as `if`, `case`, `data_bag`, `data_bag_item`, and `search`.
- About Templates – Describes how to use Embedded Ruby (ERB) templates to dynamically generate static text files, such as configuration files.
- Learning Tracks – Describes how to use Chef to manage an instance, manage a basic web app, develop and test infrastructure code, use Chef analytics, and more.
- *Learning Chef* – An introduction to Chef. Published by O'Reilly Media.
- Learning Chef code examples – Provides code examples to accompany the book *Learning Chef* published by O'Reilly Media.

OpsWorks Stacks Best Practices

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Strategies, techniques, and suggestions in this section can help you get the maximum benefit and optimal outcomes from OpsWorks Stacks.

Topics

- Best Practices: Root Device Storage for Instances
- Best Practices: Optimizing the Number of Application Servers
- Best Practices: Managing Permissions
- Best Practices: Managing and Deploying Apps and Cookbooks
- Packaging Cookbook Dependencies Locally

Best Practices: Root Device Storage for Instances

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

This topic does not apply to Windows instances, which must be Amazon Elastic Block Store-backed.

Amazon Elastic Compute Cloud (Amazon EC2) Linux instances have the following root-device storage options.

- **Instance store-backed instances** – The root device is temporary.

  If you stop the instance, the data on the root device vanishes and cannot be recovered. For more information, see Amazon EC2 Instance Store.
- **Amazon EBS-backed instances** – The root device is an Amazon EBS volume.

  If you stop the instance, the Amazon EBS volume persists. If you restart the instance, the volume is automatically remounted, restoring the instance state and any stored data. You can also mount the volume on a different instance. For more information, see Amazon Elastic Block Store (Amazon EBS).

Consider the following when deciding which root device storage option to use.

Boot Time

After the initial start, Amazon EBS instances generally restart faster.

The initial startup time is approximately the same for either storage type. Both types must perform a full setup, which includes relatively time-consuming tasks such as installing packages

from remote repositories. However, note these distinctions when you subsequently restart an instance:

- Instance store-backed instances perform the same setup tasks that they did for the initial start, including package installation.

  A restart takes about the same time as the initial start.
- Amazon EBS-back instances remount the root volume and run the Setup recipes.

  The restart is usually significantly faster than the initial start, because the Setup recipes don't have to perform tasks such as reinstalling packages that are already installed on the root volume.

Cost

Amazon EBS-backed instances are more costly:

- With an instance-store backed instance, you pay only when the instance is running.
- With Amazon EBS-backed instances, you pay for the Amazon EBS volume whether the instance is running or not.

  For more information, see Amazon EBS Pricing.

Logging

Amazon EBS-backed instances automatically retain logs:

- With instance store-backed instance, the logs disappear when the instance stops.

  You must either retrieve the logs before you stop the instance or use a service such as CloudWatch Logs to store selected logs remotely.
- With an Amazon EBS-backed instance, the logs are stored on the Amazon EBS volume.

  You can view them by restarting the instance, or by mounting the volume on another instance.

Dependencies

The two storage types have different dependencies:

- Instance-store backed instances depend on Amazon S3.

  When you start the instance, it must download the AMI from Amazon S3.
- Amazon EBS-backed instances depend on Amazon EBS.

When you start the instance, it must mount the Amazon EBS root volume.

Recommendation: If you aren't certain which storage type is best suited for your requirements, we recommend starting with Amazon EBS instances. Although you will incur a modest expense for the Amazon EBS volumes, there is less risk of unintended data loss.

Best Practices: Optimizing the Number of Application Servers

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

A production stack commonly includes multiple application servers distributed across multiple Availability Zones. However the number of incoming requests can vary substantially depending on time of day or day of the week. You could just run enough servers to handle the maximum anticipated load, but then much of the time you will end up paying for more server capacity than you need. To run your site efficiently, the recommended practice is to match the number of servers to the current request volume.

OpsWorks Stacks provides three ways to manage the number of server instances.

- 24/7 instances are started manually and run until they are manually stopped.
- Time-based instances are automatically started and stopped by OpsWorks Stacks on a user-specified schedule.
- Load-based instances are automatically started and stopped by OpsWorks Stacks when they cross a threshold for a user-specified load metric such as CPU or memory utilization.

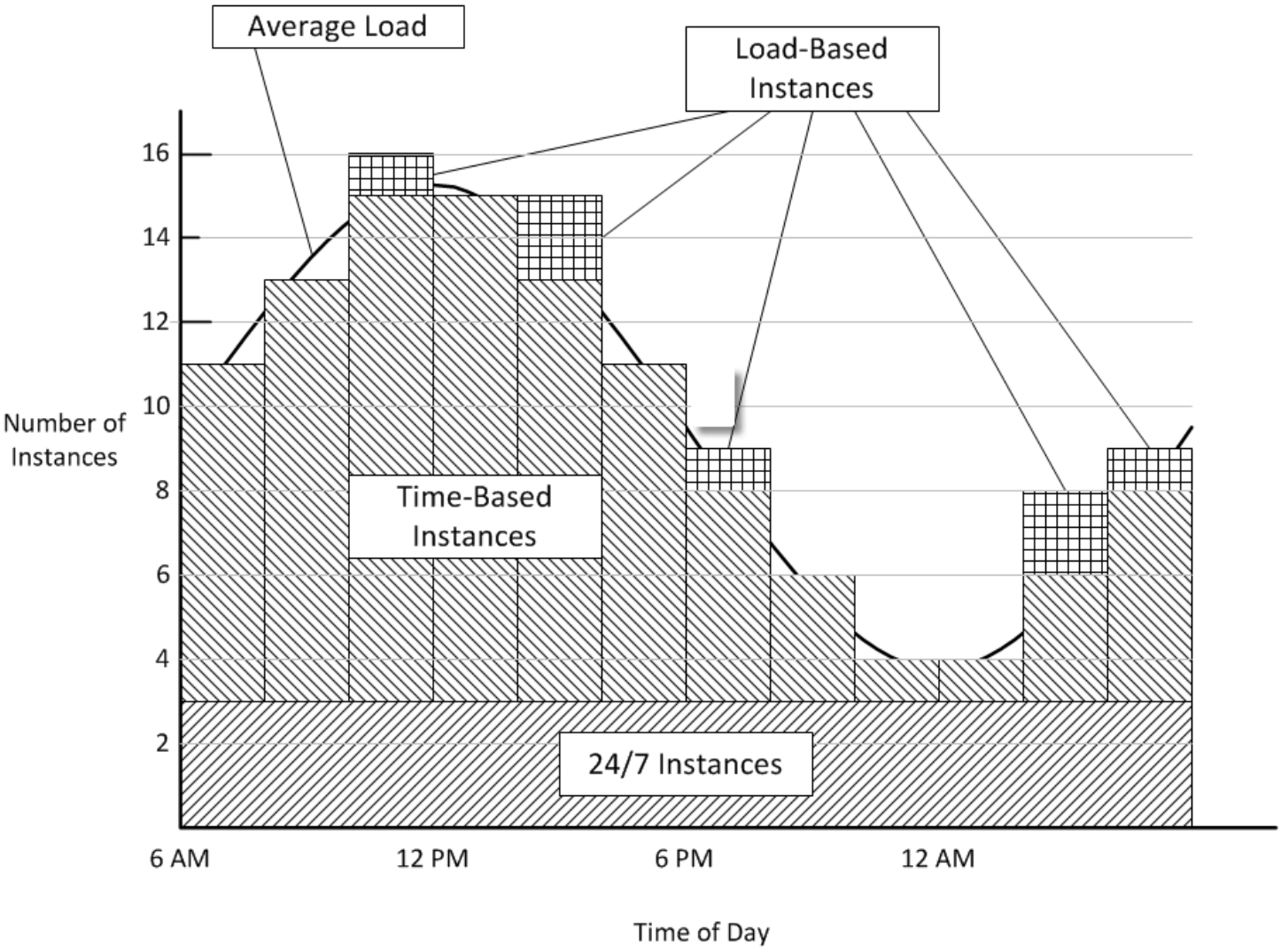
> **Note**
>
> After you have created and configured your stack's time and load-based instances, OpsWorks Stacks automatically starts and stops them based on the specified configuration.

You don't have to touch them again unless you decide to change the configuration or number of instances.

Recommendation: If you are managing stacks with more than a few application server instances, we recommend using a mix of all three instance types. The following is an example of how to manage a stack's server capacity to handle a variable daily request volume with the following characteristics.

- The average request volume varies sinusoidally over the day.
- The minimum average request volume requires five application server instances.
- The maximum average request volume requires sixteen application server instances.
- Spikes in request volume can usually be handled by one or two application server instances.

This is a convenient model for the purposes of discussion, but you can easily adapt it to any variation in request volume and also extend it to handle weekly variations. The following diagram shows how to use the three instance types to manage this request volume.



This example has the following characteristics:

- The stack has three 24/7 instances, which are always on and handle the base load.
- The stack has 12 time-based instances, which are configured to handle the average daily variation.

  One runs from 10 PM to 2 AM, two more run from 8 PM to 10 PM and 2 AM to 4 AM, and so on. For simplicity, the diagram modifies the number of time-based instances every two hours, but you can modify the number every hour if you want finer-grained control.
- The stack has enough load-based instances to handle traffic spikes that exceed what can be handled by the 24/7 and time-based instances.

  OpsWorks Stacks starts load-based instances only when the load across all of the currently running servers exceeds the specified metrics. The cost for nonrunning instances is minimal

(Amazon EBS-backed instances) or nothing (instance store-backed instances), so the recommended practice is to create enough of them to comfortably handle your maximum anticipated request volumes. For this example, the stack should have at least three load-based instances.

> **Note**
>
> Make sure you have all three instance types distributed across multiple Availability Zones to mitigate the impact of any service disruptions.

Best Practices: Managing Permissions

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You must have some form of AWS credentials to access your account's resources. The following are some general guidelines for providing access to your employees.

- First and foremost, we recommend that you do not use your account's root credentials to access AWS resources.

  Instead, create IAM Identities for your employees and add permissions that provide appropriate access. Each employee can then use their credentials to access resources.
- Employees should have permissions to access only those resources that they need to perform their jobs.

  For example, application developers need to access only the stacks that run their applications.
- Employees should have permissions to use only those actions that they need to perform their jobs.

An application developer might need full permissions for a development stack and permissions to deploy their apps to the corresponding production stack. They probably do not need permissions to start or stop instances on the production stack, create or delete layers, and so on.

For more general information on managing permissions, see AWS Security Credentials.

You can use OpsWorks Stacks or IAM to manage user permissions. Note that the two options are not mutually exclusive; it is sometimes desirable to use both.

OpsWorks Stacks Permissions Management

Each stack has a **Permissions** page that you can use to grant users permission to access the stack and specify what actions they can take. You specify a user's permissions by setting one of the following permissions levels. Each level represents an IAM policy that grants permissions for a standard set of actions.

- **Deny** denies permission to interact with the stack in any way.
- **Show** grants permissions view the stack configuration but not modify the stack state in any way.
- **Deploy** includes the **Show** permissions and also grants the user permissions to deploy apps.
- **Manage** includes the **Deploy** permissions and also allows the user to perform a variety of stack management actions, such as creating or deleting instances and layers.

> **Note**
>
> The **Manage** permissions level does not grant permissions for a small number of high-level OpsWorks Stacks actions, including creating or cloning stacks. You must use an IAM policy to grant those permissions.

In addition to setting permissions levels, you can also use a stack's **Permissions** page to specify whether users have SSH/RDP and sudo/admin privileges on the stack's instances. For more information about OpsWorks Stacks permissions management, see Granting Per-Stack Permissions. For more information about managing SSH access, see Managing SSH Access.

IAM Permissions Management

With IAM permissions management, you use the IAM console, API, or CLI to attach a JSON-formatted policy to a user that explicitly specifies their permissions. For more information about IAM permissions management, see What is IAM?.

Recommendation: Start with OpsWorks Stacks **Permissions** management. If you need to fine tune a user's permissions, or grant a user permissions that aren't included in the **Manage** permissions levels, you can combine the two approaches. OpsWorks Stacks then evaluates both policies to determine the user's permissions.

> **Important**
>
> If a user has multiple policies with conflicting permissions, denial always wins. For example, suppose that you attach an IAM policy to a user that allows access to a particular stack but also use the stack's **Permissions** page to assign the user a **Deny** permissions level. The **Deny** permissions level takes precedence, and the user will not be able to access the stack. For more information, see IAM policy evaluation logic.

For example, suppose you want a user to be able to perform most operations on a stack, except for adding or deleting layers.

- Specify a **Manage** permissions level, which allows the user to perform most stack management actions, including creating and deleting layers.
- Attach the following customer-managed policy to the user, which denies permissions to use the CreateLayer and DeleteLayer actions on that stack. You identify the stack by its *Amazon Resource Name (ARN)*, which can be found on the stack's **Settings** page.

  JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "opsworks:CreateLayer",
```

```
        "opsworks:DeleteLayer"
      ],
      "Resource": "arn:aws:opsworks:*:*:stack/2f18b4cb-4de5-4429-a149-
ff7da9f0d8ee/"
    }
  ]
}
```

For more information, including example policies, see Managing OpsWorks Stacks Permissions by Attaching an IAM Policy.

Note

Another way to use IAM policy is to set a condition that limits stack access to employees with a specified IP address or address range. For example, to ensure that employees access stacks only from inside your corporate firewall, set a condition that limits access to your corporate IP address range. For more information, see Conditions.

Best Practices: Managing and Deploying Apps and Cookbooks

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks deploys apps and cookbooks to each new instance from a remote repository. During an instance's lifetime, you often must update the apps or cookbooks on the stack's online instances to add features, fix bugs, and so on. There are a variety of ways to manage a stack's apps and cookbooks, but the approach you use should satisfy the following general requirements:

- All production stack instances should have the same application and custom cookbook code, with limited exceptions for purposes such as A/B testing.
- Deploying an update should not interrupt the site's operation, even if something goes wrong.

This section describes recommended practices for managing and deploying apps and custom cookbooks.

Topics

- Maintaining Consistency
- Deploying Code to Online Instances

Maintaining Consistency

In general, you need to maintain tight control over the app or cookbook code that runs on your production stack. Typically, all instances should run the currently approved version of the code. Exceptions occur when updating your apps or cookbooks, as described later, and when accommodating special cases, such as performing A/B testing.

App and cookbook code is deployed from a specified source repository to your stack's instances in two ways:

- When you start an instance, OpsWorks Stacks automatically deploys the current app and cookbook code to the instance.
- For online instances, you must manually deploy the current app or cookbook code by running a Deploy command (for apps) or an Update Custom Cookbooks command (for cookbooks).

Because there are two deployment mechanisms, it's critical that you manage your source code carefully to avoid unintentionally running different code on different instances. For example, if you deploy apps or cookbooks from a Git master branch, OpsWorks Stacks deploys what is in that branch at the time. If you update the code in the master branch and then start a new instance, that instance will have a more recent version of the code than older instances. The more recent version might not even be approved for production.

Recommendation: Amazon S3 Archives

To ensure that all your instances have the approved code version, we recommend deploying your apps and cookbooks from an Amazon Simple Storage Service (Amazon S3) archive. This guarantees that the code is a static artifact—a .zip or other archive file—that must be explicitly updated. In addition, Amazon S3 is highly reliable, so you will rarely, if ever, be unable to access the archive. To further ensure consistency, explicitly version each archive file by using a naming

convention or by using Amazon S3 versioning, which provides an audit trail and an easy way to revert to an earlier version.

For example, you could create a deployment pipeline using a tool such as Jenkins. After the code that you want to deploy has been committed and tested, create an archive file and upload it to Amazon S3. All app deployments or cookbook updates will install the code in that archive file and every instance will have the same code.

Recommendation: Git or Subversion Repositories

If you prefer to use a Git or Subversion repository, don't deploy from the master branch. Instead, tag the approved version and specify that version as the app or cookbook source.

Deploying Code to Online Instances

OpsWorks Stacks does not automatically deploy updated code to online instances. You must perform that operation manually, which poses the following challenges:

- Deploying the update efficiently without compromising the site's ability to handle customer requests during the deployment process.
- Handling an unsuccessful deployment, either because of problems with the deployed app or cookbooks or problems with the deployment process itself.

The simplest approach is to run a default Deploy command (for apps) or Update Custom Cookbooks command (for cookbooks), which deploys the update to every instance concurrently. This approach is simple and fast, but there is no margin for error. If the deployment fails or the updated code has any issues, every instance in your production stack could be affected, potentially disrupting or disabling your site until you can fix the problem or roll back to the previous version.

Recommendation: Use a robust deployment strategy, which allows instances running the old version of code to continue handling requests until you have verified that deployment was successful and can confidently transfer all incoming traffic to the new version.

The following sections provide two examples of robust deployment strategies, followed by a discussion of how to manage a backend database during deployment. For brevity, they describe app updates, but you can use similar strategies for cookbooks.

Topics

- Using a Rolling Deployment

- Using Separate Stacks
- Managing a Backend Database

Using a Rolling Deployment

A rolling deployment updates an application on a stack's online application server instances in multiple phases. With each phase, you update a subset of the online instances and verify that the update is successful before starting the next phase. If you encounter problems, the instances that are still running the old app version can continue to handle incoming traffic until you resolve the issues.

The following example assumes that you are using the recommended practice of distributing your stack's application server instances across multiple Availability Zones.

To perform a rolling deployment

1. On the Deploy App page, choose **Advanced**, choose a single application server instance, and deploy the app to that instance.

   If you want to be cautious, you can remove the instance from the load balancer before deploying the app. This ensures that users won't encounter the updated application until you have verified that it is working correctly. If you use Elastic Load Balancing, remove the instance from the load balancer by using the Elastic Load Balancing console, CLI, or an SDK.
2. Verify that the updated app is working correctly and that the instance has acceptable performance metrics.

   If you removed the instance from an Elastic Load Balancing load balancer, use the Elastic Load Balancing console, CLI, or an SDK to restore it. The updated app version is now handling user requests.
3. Deploy the update to the remainder of the instances in the Availability Zone and verify that they are working correctly and have acceptable metrics.
4. Repeat step 3 for the stack's other Availability Zones, one zone at a time. If you want to be especially cautious, repeat steps 1 – 3.

> **Note**
>
> If you use an Elastic Load Balancing load balancer, you can use its health check to verify that the deployment was successful. However, set the ping path to an application that checks dependencies and verifies that everything is working correctly, not a static file that simply confirms that the application server is running.

Using Separate Stacks

Another approach to managing applications is to use a separate stack for each phase of the application's lifecycle. The different stacks are sometimes referred to as environments. This arrangement allows you to do development and testing on stacks that are not publicly accessible. When you are ready to deploy an update, switch user traffic from the stack that hosts the current application version to the stack that hosts the updated version.

Topics

- Using Development, Staging, and Production Stacks
- Using a Blue-Green Deployment Strategy

Using Development, Staging, and Production Stacks

The most common approach uses the following stacks.

Development Stack

Use a development stack for tasks such as implementing new features or fixing bugs. A development stack is essentially a prototype production stack, with the same layers, apps, resources, and so on that are included on your production stack. Because the development stack usually does not have to handle the same load as the production stack, you typically can use fewer or smaller instances.

Development stacks are not public facing; you control access as follows:

- Restrict network access by configuring the application server's or load balancer's security group inbound rules to accept incoming requests only from specified IP addresses or address ranges.

  For example, limit HTTP, HTTPS, and SSH access to addresses in your corporate address range.

- Control access to OpsWorks Stacks stack management functionality by using the stack's Permissions page.

  For example, grant a Manage permissions level to the development team, and Show permissions to all other employees.

Staging Stack

Use a staging stack to test and finalize candidates for an updated production stack. When you have completed development, create a staging stack by cloning the development stack. Then run your test suite on the staging stack and deploy updates to that stack to fix issues that arise.

Staging stacks also are not public facing; you control stack and network access the same way you do for the development stack. Note that when you clone a development stack to create a staging stack, you can clone the permissions granted by OpsWorks Stacks permissions management. However, cloning does not affect permissions granted by users' IAM policies. You must use the IAM console, CLI, or an SDK to modify those permissions. For more information, see Managing User Permissions.

Production Stack

The production stack is the public-facing stack that supports your current application. When the staging stack has passed testing, you promote it to production and retire the old production stack. For an example of how to do this, see Using a Blue-Green Deployment Strategy.

> **Note**
>
> Instead of using the OpsWorks Stacks console to create stacks manually, create an AWS CloudFormation template for each stack. This approach has the following advantages:
>
> - Speed and convenience – When you launch the template, AWS CloudFormation automatically creates the stack, including all the required instances.
> - Consistency – Store the template for each stack in your source repository to ensure that developers use the same stack for the same purpose.

Using a Blue-Green Deployment Strategy

A *blue-green* deployment strategy is one common way to efficiently use separate stacks to deploy an application update to production.

- The blue environment is the production stack, which hosts the current application.
- The green environment is the staging stack, which hosts the updated application.

When you are ready to deploy the updated app to production, you switch user traffic from the blue stack to the green stack, which becomes the new production stack. You then retire the old blue stack.

The following example describes how to perform a blue-green deployment with OpsWorks Stacks stacks, in conjunction with Route 53 and a pool of Elastic Load Balancing load balancers. Prior to making the switch, you should ensure the following:

- The application update on the green stack has passed testing and is ready for production.
- The green stack is identical to the blue stack except that it includes the updated app and is not public facing.

  Both stacks have the same permissions, the same number and type of instances in each layer, the same time-based and load-based configuration, and so on.
- All of the green stack's 24/7 instances and scheduled time-based instances are online.
- You have a pool of Elastic Load Balancing load balancers that can be dynamically attached to a layer in either stack and can be pre-warmed to handle the expected traffic volume.
- You have used the Route 53 weighted routing feature to create a record set in a hosted zone that includes your pooled load balancers.
- You have assigned a nonzero weight to the load balancer that is attached to your blue stack's application server layer and zero weight to the unused load balancers. This ensures that the blue stack's load balancer handles all incoming traffic.

To switch users to the green stack

1. Attach one of the pool's unused load balancers to the green stack's application server layer. In some scenarios, such as when you expect flash traffic, or if you cannot configure a load test to gradually increase traffic, pre-warm the load balancer to handle the expected traffic.
2. After all of the green stack's instances have passed the Elastic Load Balancing health check, change the weights in the Route 53 record set so that the green stack's load balancer has a nonzero weight and the blue stack's load balancer has a correspondingly reduced weight. We recommend that you start by having the green stack handle a small percentage of requests, perhaps 5%, with the blue stack handling the rest. You now have two production stacks, with

the green stack handling some of the incoming requests and the blue stack handling the remainder.

3. Monitor the green stack's performance metrics. If they are acceptable, increase the green stack's weight so that it handles perhaps 10% of the incoming traffic.
4. Repeat Step 3 until the green stack is handling approximately half of the incoming traffic. Any issues should have surfaced by this point, so if the green stack is performing acceptably, you can complete the process by reducing the blue stack's weight to zero. The green stack is now the new blue stack and is handling all incoming traffic.
5. Detach the load balancer from the old blue stack's application server layer and return it to the pool.
6. Although the old blue stack is no longer handling user requests, we recommend retaining it for a while in case there are problems with the new blue stack. In that case, you can roll back the update by reversing the procedure to direct incoming traffic back to the old blue stack. When you are confident that the new blue stack is operating acceptably, shut down the old blue stack.

Managing a Backend Database

If your application depends on a backend database, you will need to transition from the old application to the new. OpsWorks Stacks supports the following database options.

Amazon RDS Layer

With an Amazon Relational Database Service (Amazon RDS) layer, you create the RDS DB instance separately and then register it with your stack. You can register an RDS DB instance with only one stack at a time, but you can switch an RDS DB instance from one stack to another.

OpsWorks Stacks installs a file with the connection data on your application servers in a format that easily can be used by your application. OpsWorks Stacks also adds the database connection information to the stack configuration and deployment attributes, which can be accessed by recipes. You also can use JSON to provide connection data to applications. For more information, see Connecting to a Database.

Updating an application that depends on a database poses two basic challenges:

- Ensuring that every transaction is properly recorded during the transition while also avoiding race conditions between the new and old application versions.

- Performing the transition in a way that limits the impact on your site's performance and minimizes or eliminates downtime.

When you use the deployment strategies described in this topic, you can't simply detach the database from the old application and reattach it to the new one. Both versions of the application run in parallel during the transition and must have access to the same data. The following describes two approaches to managing the transition, both of which have advantages and challenges.

Approach 1: Have both applications connect to the same database

Advantages

- There is no downtime during the transition.

  One application gradually stops accessing the database while the other gradually takes over.
- You don't have to synchronize data between two databases.

Challenges

- Both applications access the same database, so you must manage access to prevent data loss or corruption.
- If you need to migrate to a new database schema, the old application version must be able to use the new schema.

If you are using separate stacks, this approach is probably best suited to Amazon RDS because the instance is not permanently tied to a particular stack and can be accessed by applications running on different stacks. However, you can't register an RDS DB instance with more than one stack at a time, so you must provide connection data to both applications, for example by using JSON. For more information, see Using a Custom Recipe.

If you use a rolling upgrade, the old and new application versions are hosted on the same stack, so you can use either an Amazon RDS or MySQL layer.

Approach 2: Provide each application version with its own database

Advantages

- Each version has its own database, so the schemas don't have to be compatible.

Challenges

- Synchronizing the data between the two databases during the transition without losing or corrupting data.

- Ensuring that your synchronization procedure doesn't cause significant downtime or significantly degrade the site's performance.

If you are using separate stacks, each one has its own database. If you are using a rolling deployment, you can attach two databases to the stack, one for each application. If the old and updated applications do not have compatible database schemas, this approach is better.

Recommendation: In general, we recommend using an Amazon RDS layer as an application's backend database because it is more flexible and can be used for any transition scenario. For more information about how to handle transitions, see the Amazon RDS User Guide.

Packaging Cookbook Dependencies Locally

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can use Berkshelf to package your cookbook dependencies locally, upload the package to Amazon S3, and modify your stack to use the package on Amazon S3 as a cookbook source. Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

The following walkthroughs describe how to pre-package your cookbooks and their dependencies into a .zip file, and then use the .zip file as your cookbook source for Linux instances in OpsWorks Stacks. The first walkthrough describes how to package one cookbook. The second walkthrough describes how to package multiple cookbooks.

Before you begin, install the Chef Development Kit (also known as Chef DK), which is an assortment of tools built by the Chef community. You will need this to use the `chef` command-line tool.

Packaging Dependencies Locally in Chef 12

In Chef 12 Linux, Berkshelf is no longer installed by default on stack instances. We recommend that you install and use Berkshelf on a local development computer to package your cookbook dependencies locally. Upload your package, with the dependencies included, to Amazon S3. Finally, modify your Chef 12 Linux stack to use the uploaded package as a cookbook source. Be aware of the following differences when you are packaging cookbooks in Chef 12.

1. On the local computer, create a cookbook by running the `chef` command line tool.

   ```
   chef generate cookbook "server-app"
   ```

   This command creates a cookbook, a Berksfile, a `metadata.rb` file, and a recipe directory, and places them in a folder that has the same name as the cookbook. The following example shows the structure of what is created.

   ```
   server-app <-- the cookbook you've just created
       ### Berksfile
       ### metadata.rb
       ### recipes
   ```

2. In a text editor, edit the Berksfile to point to cookbooks on which the `server-app` cookbook will depend. In our example, we want `server-app` to depend on the `java` cookbook from the Chef Supermarket. We are specifying the version 1.50.0 or newer *minor* version, but you can enter any published version in the single quotation marks. Save your changes and close the file.

   ```
   source 'https://supermarket.chef.io'
   cookbook 'java', '~> 1.50.0'
   ```

3. Edit the `metadata.rb` file to add the dependency. Save your changes and close the file.

   ```
   depends 'java' , '~> 1.50.0'
   ```

4. Change to the `server-app` cookbook directory that Chef created for you, and then run the `package` command to create a `tar` file of the cookbook. If you are packaging multiple cookbooks, you want to run this command at the root directory in which all cookbooks are stored. To package a single cookbook, run this command at the cookbook directory level. In this example, we run this command in the `server-app` directory.

```
berks package cookbooks.tar.gz
```

The output resembles the following. The `tar.gz` file is created in your local directory.

```
Cookbook(s) packaged to /Users/username/tmp/berks/cookbooks.tar.gz
```

5. In the AWS CLI, upload the package you just created to Amazon S3. Make a note of the new URL of the cookbook package after you have uploaded it to S3; you'll need this URL for your stack settings.

```
aws s3 cp cookbooks.tar.gz s3://bucket-name/
```

The output resembles the following.

```
upload: ./cookbooks.tar.gz to s3://bucket-name/cookbooks.tar.gz
```

6. In OpsWorks Stacks, modify your stack to use the package that you uploaded as the cookbook source.

   a. Set the **Use custom Chef cookbooks** setting to **Yes**.

   b. Set **Repository type** to **S3 Archive**.

   c. In **Repository URL**, paste the URL of the cookbook package that you uploaded in step 5.

   Save your stack changes.

Packaging Dependencies Locally for One Cookbook

1. On the local computer, create a cookbook by using the chef command line tool:

```
chef generate cookbook "server-app"
```

This command creates a cookbook and a Berksfile, and places them in a folder that has the same name as the cookbook.

2. Change to the cookbook directory that Chef created for you, and then package everything by running the following command:

```
berks package cookbooks.tar.gz
```

The output looks like this:

```
Cookbook(s) packaged to /Users/username/tmp/berks/cookbooks.tar.gz
```

3. In the AWS CLI, upload the package you just created to Amazon S3:

```
aws s3 cp cookbooks.tar.gz s3://bucket-name/
```

The output looks like this:

```
upload: ./cookbooks.tar.gz to s3://bucket-name/cookbooks.tar.gz
```

4. In OpsWorks Stacks, modify your stack to use the package that you uploaded as the cookbook source.

Packaging Dependencies Locally for Multiple Cookbooks

This example creates two cookbooks and packages the dependencies for them.

1. On the local computer, run the following `chef` commands to generate two cookbooks:

```
chef generate cookbook "server-app"
chef generate cookbook "server-utils"
```

In this example, the server-app cookbook performs Java configurations, so we need to add a dependency on Java.

2. Edit `server-app/metadata.rb` to add a dependency on the community Java cookbook:

```
maintainer "The Authors"
maintainer_email "you@example.com"
license "all_rights"
description "Installs/Configures server-app"
long_description "Installs/Configures server-app"
version "0.1.0"
depends "java"
```

3. Tell Berkshelf what to package by editing the Berksfile file in the cookbook root directory as follows:

```
source "https://supermarket.chef.io"
cookbook "server-app", path: "./server-app"
cookbook "server-utils", path: "./server-utils"
```

   Your file structure now looks like this:

```
..
   ### Berksfile
   ### server-app
   ### server-utils
```

4. Finally, create a zip package, upload it to Amazon S3, and modify your OpsWorks Stacks stack to use the new cookbook source. To do this, follow steps 2 through 4 in Packaging Dependencies Locally for One Cookbook.

Additional resources

For more information about packaging cookbook dependencies, see the following.

- How to Package Cookbook Dependencies Locally with Berkshelf on the AWS DevOps Blog
- Linux Chef 12 with Berkshelf on the OpsWorks forums
- Berkshelf in Chef 12 on the OpsWorks forums
- Installing Custom Cookbooks in this guide
- Cookbook Repositories in this guide

Stacks

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The stack is the top-level OpsWorks Stacks entity. It represents a set of instances that you want to manage collectively, typically because they have a common purpose such as serving PHP applications. In addition to serving as a container, a stack handles tasks that apply to the group of instances as a whole, such as managing applications and cookbooks.

For example, a stack whose purpose is to serve web applications might look something like the following:

- A set of application server instances, each of which handles a portion of the incoming traffic.
- A load balancer instance, which takes incoming traffic and distributes it across the application servers.
- A database instance, which serves as a back-end data store for the application servers.

A common practice is to have multiple stacks that represent different environments. A typical set of stacks consists of:

- A development stack to be used by developers to add features, fix bugs, and perform other development and maintenance tasks.
- A staging stack to verify updates or fixes before exposing them publicly.
- A production stack, which is the public-facing version that handles incoming requests from users.

This section describes the basics of working with stacks.

Topics

- Migrating stacks from Amazon EC2-Classic to a VPC
- Create a New Stack
- Running a Stack in a VPC
- Update a Stack
- Clone a Stack
- Run OpsWorks Stacks Stack Commands
- Using Custom JSON

- Delete a Stack

Migrating stacks from Amazon EC2-Classic to a VPC

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This topic describes how to migrate an AWS OpsWorks Stacks stack from the Amazon EC2 Classic network platform to an Amazon Virtual Private Cloud (Amazon VPC) network.

If you created your AWS account before 2013-12-04, you might have support for EC2-Classic in some AWS Regions. Some Amazon EC2 resources and features, such as enhanced networking and newer instance types, require a virtual private cloud (VPC). Some resources can be shared between EC2-Classic and a VPC, while some can't. To avoid disruptions to your service, we recommend that you migrate your AWS OpsWorks Stacks stacks to a VPC.

Topics

- Prerequisites
- Migrate an AWS OpsWorks Stacks stack to a VPC
- See also

Prerequisites

Before you begin, you must have a VPC that meets AWS OpsWorks Stacks configuration requirements. To configure private subnets in your VPC for AWS OpsWorks Stacks, see Running a Stack in a VPC in this guide. You can create a custom VPC by using the Amazon VPC management console. For more information, see Amazon VPC console wizard configurations and VPCs and subnets in the *Amazon Virtual Private Cloud User Guide*.

To continue with migration, you'll need the VPC ID and the subnet ID that you want to use.

Migrate an AWS OpsWorks Stacks stack to a VPC

First, clone an existing EC2-Classic stack by using the AWS OpsWorks Stacks console or API. Then, move the existing stack's resources to the new stack. Start the new instances in the cloned stack, and deploy apps. Verify that the new stack is working. Finally, delete the EC2-Classic resources from the EC2-Classic stack, and then delete the old stack.

1. Clone your existing EC2-Classic stack into your VPC. Cloning the stack copies stack settings, layers, apps, users, and user permissions to the new stack. For more information about how to clone a stack, see Clone a Stack in this guide.

   You can also clone a stack by using the AWS OpsWorks Stacks API. When you clone a stack by using the AWS CLI or AWS SDKs, set the value of the `VpcId` parameter to the ID of the VPC that you created in Prerequisites. For more information, see **CloneStack** in the *AWS OpsWorks Stacks API Reference*.

2. Create new instances in the layers of the cloned stack. Be sure to specify the ID of the subnet you created in Prerequisites. For more information about how to create instances in a stack, see Adding an Instance to a Layer in this guide.

3. Migrate your classic resources, such as EC2 security groups, Elastic Load Balancing load balancers, and Elastic IP addresses to your VPC, and then associate them with the cloned stack. For more information, see Migrate your resources to a VPC in the *Amazon EC2 User Guide*.

4. Register Amazon EBS volumes and Amazon RDS instances with the cloned stack. For more information about registering resources with a stack, see Registering Resources with a Stack in this guide.

   Amazon EBS volumes aren't associated with a VPC, and you can use them across instances in both EC2-Classic stacks and stacks in a VPC. You can register Amazon RDS instances in EC2-Classic with both EC2-Classic stacks and stacks in a VPC.

5. Start instances in the cloned stack, and then move a small percentage of your workloads to the cloned stack. For example, move a small percentage of traffic to the Elastic Load Balancing load balancers in the cloned stack. If you are using Amazon Route 53, see Routing traffic to an ELB load balancer in the *Amazon Route 53 Developer Guide*.

   Route only a small percentage of traffic until you are sure that the new stack is functional and supports your applications. Let the new stack work with a small percentage of traffic for a trial period, such as a week. After you verify that the new stack is working, route remaining traffic to the stack.

6. After you are sure the cloned stack is working, move the remainder of your production traffic or workloads to the cloned stack. You can now stop instances in the EC2-Classic stack. We recommend that you keep the old stack available for several weeks, so you can move workloads back to the old stack if any issues occur with the new stack in the weeks after the migration.
7. When the new stack has been working for several weeks, delete instances in the EC2-Classic stack. For more information about how to delete instances, see Deleting OpsWorks Stacks Instances in this guide.

   > **Important**
   >
   > Do not use the Amazon EC2 console or API to stop or delete OpsWorks instances.

8. Delete apps in the EC2-Classic stack. For more information about how to delete apps, see To delete the app from the stack in this guide.
9. Delete the EC2-Classic stack. For more information about how to delete a stack, see Delete a Stack in this guide.

See also

- Migrating from EC2-Classic to a VPC
- Debugging and Troubleshooting Guide
- Running a Stack in a VPC

Create a New Stack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

To create a new stack, on the OpsWorks Stacks dashboard, click **Add stack**. You can then use the **Add Stack** page to configure the stack. When you are finished, click **Add Stack**.

Topics

- Choose the Type of Stack to Create
- Basic Options
- Advanced Options

Choose the Type of Stack to Create

Before you create a stack, you must decide the type of stack that you want to create. For help, see the following table.

If you want to create...	Create this type of stack if you want to...	To learn how, follow these instructions:
A sample stack	Explore the basics of AWS OpsWorks with a Linux-based Chef 12 stack and a sample Node.js app.	Getting Started: Sample
A Linux-based Chef 12 stack	Create a Linux-based stack that uses the latest version of Chef that AWS OpsWorks supports. Choose this option if you are an advanced Chef user who would like to benefit from the large selection of community cookbooks or write your own custom cookbooks. For more information, see Chef 12 Linux.	Getting Started: Linux
A Windows-based Chef 12.2 stack	Create a Windows-based stack.	Getting Started: Windows

If you want to create...	Create this type of stack if you want to...	To learn how, follow these instructions:
A Linux-based Chef 11.10 stack	Create this stack if your organization requires the use of Chef 11.10 with Linux for backward compatibility.	Getting Started with Chef 11 Linux Stacks

Basic Options

The **Add Stack** page has the following basic options.

Stack name

(Required) A name that is used to identify the stack in the OpsWorks Stacks console. The name does not need to be unique. OpsWorks Stacks also generates a stack ID, which is a GUID that uniquely identifies the stack. For example, with AWS CLI commands such as update-stack, you use the stack ID to identify the particular stack. After you have created a stack, you can find it's ID by choosing **Stack** in the navigation pane and then choosing **Stack Settings**. The ID is labelled **OpsWorks ID**.

Region

(Required) The AWS region where the instances will be launched.

VPC

(Optional) The ID of the VPC that the stack is to be launched into. All instances will be launched into this VPC, and you cannot change the ID later.

- If your account supports EC2 Classic, you can specify **No VPC** (the default value) if you don't want to use a VPC.

  For more information about EC2 Classic, see Supported Platforms.

- If your account does not support EC2 Classic, you must specify a VPC.

  The default setting is **Default VPC**, which combines the ease of use of EC2 Classic with the benefits of VPC networking features. If you want to run your stack in a regular VPC, you must create it by using the VPC console, API, or CLI. For more information on how to create a VPC for an OpsWorks Stacks stack, see Running a Stack in a VPC. For general information, see Amazon Virtual Private Cloud.

Default Availability Zone/Default subnet

(Optional) This setting depends on whether you are creating your stack in a VPC:

- If your account supports EC2 Classic and you set **VPC** to **No VPC**, this setting is labeled **Default Availability Zone**, which specifies the default AWS Availability Zone where the instances will be launched.
- If your account does not support EC2 Classic or you choose to specify a VPC, this field is labeled **Default subnet**, which specifies the default subnet where the instances will be launched. You can launch an instance in other subnets by overriding this value when you create the instance. Each subnet is associated with one Availability Zone.

You can have OpsWorks Stacks launch an instance in a different Availability Zone or subnet by overriding this setting when you create the instance.

For more information about how to run a stack in a VPC, see Running a Stack in a VPC.

Default operating system

(Optional) The operating system that is installed by default on each instance. You have the following options:

- One of the built-in Linux operating systems.
- Microsoft Windows Server 2012 R2.
- A custom AMI based on one of the supported operating systems.

  If you select **Use custom AMI**, the operating system is determined by a custom AMI that you specify when you create instances. For more information, see Using Custom AMIs.

For more information on the available operating systems, see OpsWorks Stacks operating systems.

> **Note**
>
> You can override the default operating system when you create an instance. However, you cannot override a Linux operating system to specify Windows, or Windows to specify a Linux operating system.

Default SSH key

(Optional) An Amazon EC2 key pair from the stack's region. The default value is none. If you specify a key pair, OpsWorks Stacks installs the public key on the instance.

- With Linux instances, you can use the private key with an SSH client to log in to the stack's instances.

  For more information, see Logging In with SSH.

- With Windows instances, you can use the private key with the Amazon EC2 console or CLI to retrieve an instance's Administrator password.

  You can then use that password with an RDP client to log in to the instance as Administrator. For more information, see Logging In with RDP.

For more information on how to manage SSH keys, see Managing SSH Access.

> **Note**
>
> You can override this setting by specifying a different key pair, or no key pair, when you create an instance.

Chef version

This shows the Chef version that you have chosen.

For more information on Chef versions, see Chef Versions.

Use custom Chef cookbooks

Whether to install your custom Chef cookbooks on the stack's instances.

For Chef 12, the default setting is **Yes**. For Chef 11, The default setting is **No**. The **Yes** option displays several additional settings that provide OpsWorks Stacks with the information it needs to deploy the custom cookbooks from their repository to the stack's instances, such as the repository URL. The details depend on which repository you use for your cookbooks. For more information, see Installing Custom Cookbooks.

Stack color

(Optional) The hue used to represent the stack on the OpsWorks Stacks console. You can use different colors for different stacks to help distinguish, for example, among development, staging, and production stacks.

Stack tags

You can apply tags at the stack and layer level. When you create a tag, you are applying the tag to every resource within the tagged structure. For example, if you apply a tag to a stack, you are applying the tag to every layer, and within each layer, to every instance, Amazon EBS volume, or Elastic Load Balancing load balancer in the layer. For more information about how to activate your tags and use them to track and manage the costs of your OpsWorks Stacks resources, see Using Cost Allocation Tags and Activating User-Defined Cost Allocation Tags in the *Billing and Cost Management User Guide*. For more information about tagging in OpsWorks Stacks, see Tags.

Advanced Options

For advanced settings, click **Advanced >>** to display the **Advanced options** and **Security** sections.

The **Advanced options** section has the following options:

Default root device type

Determines the type of storage to be used for the instance's root volume. For more information, see Storage.

- Linux stacks use an Amazon EBS-backed root volume by default but you can also specify an instance store-backed root volume.
- Windows stacks must use an Amazon EBS-backed root volume.

IAM role

(Optional) The stack's AWS Identity and Access Management (IAM) role, which OpsWorks Stacks uses to interact with AWS on your behalf.

Default IAM instance profile

(Optional) The default IAM role to be associated with the stack's Amazon EC2 instances. This role grants permissions to applications running on the stack's instances to access AWS resources such as S3 buckets.

- To grant specific permissions to applications, choose an existing instance profile (role) that has the appropriate policies.
- Initially, the profile's role grants no permissions, but you can use the IAM console, API, or CLI to attach appropriate policies. For more information, see Specifying Permissions for Apps Running on EC2 instances.

API endpoint region

This setting takes its value from the region that you choose in the stack's basic settings. You can choose from the following regional endpoints.

- US East (N. Virginia) Region
- US East (Ohio) Region
- US West (Oregon) Region
- US West (N. California) Region
- Canada (Central) Region (API only; not available for stacks created in the AWS Management Console
- Asia Pacific (Mumbai) Region
- Asia Pacific (Singapore) Region
- Asia Pacific (Sydney) Region
- Asia Pacific (Tokyo) Region
- Asia Pacific (Seoul) Region
- Europe (Frankfurt) Region
- Europe (Ireland) Region
- Europe (London) Region
- Europe (Paris) Region
- South America (São Paulo) Region

Stacks that are created in one API endpoint are not available in another API endpoint. Because OpsWorks Stacks users are also region-specific, if you want OpsWorks Stacks users in one of these endpoint regions to manage stacks in another endpoint region, you must import the users to the endpoint with which the stacks are associated. For more information about importing users, see Importing Users into OpsWorks Stacks.

Hostname theme

(Optional) A string that is used to generate a default hostname for each instance. The default value is **Layer Dependent**, which uses the short name of the instance's layer and appends a unique number to each instance. For example, the role-dependent **Load Balancer** theme root is "lb". The first instance you add to the layer is named "lb1", the second "lb2", and so on.

OpsWorks Agent version

(Optional) Whether to automatically update the OpsWorks Stacks agent when a new version is available, or use a specified agent version and manually update it. This feature is available on Chef 11.10 and Chef 12 stacks. The default setting is **Manual update**, set to the latest agent version.

OpsWorks Stacks installs an agent on each instance that communicates with the service and handles tasks such as initiating Chef runs in response to lifecycle events. This agent is regularly updated. You have two options for specifying the agent version for your stack.

- **Auto-update** – OpsWorks Stacks automatically installs each new agent version on the stack's instances as soon as the update is available.
- **Manual update** – OpsWorks Stacks installs the specified agent version on the stack's instances.

  OpsWorks Stacks posts a message on the stack page when a new agent version is available, but does not update the stack's instances. To update the agent, you must manually update the stack settings to specify a new agent version and OpsWorks Stacks will then update the stack's instances.

You can override the default **OpsWorks Agent Version** setting for a particular instance by updating its configuration. In that case, the instance's setting takes precedence. For example, suppose that the default setting is **Auto-update** but you specify **Manual update** for a particular instance. When OpsWorks Stacks releases a new agent version, it will automatically update all of the stack's instances except for the one that is set to **Manual update**. To install a new agent version on that instance, you must manually update its configuration and specify a new version.

> **Note**
>
> The console displays abbreviated agent version numbers. To see full version numbers, call the AWS CLI describe-agent-versions command or the equivalent API or SDK methods. They return the full version numbers for the available agent versions.

Custom JSON

(Optional) One or more custom attributes, formatted as a JSON structure. These attributes are merged into the stack configuration and deployment attributes that are installed on every instance and can be used by recipes. You can use custom JSON, for example, to customize configuration settings by overriding the built-in attributes that specify the default settings. For more information, see Using Custom JSON.

Security has one option, **Use OpsWorks security groups**, which allows you to specify whether to associate the OpsWorks Stacks built-in security groups with the stack's layers.

OpsWorks Stacks provides a standard set of built-in security groups—one for each layer— which are associated with layers by default. **Use OpsWorks security groups** allows you to instead provide your own custom security groups. For more information, see Using Security Groups.

Use OpsWorks security groups has the following settings:

- **Yes** - OpsWorks Stacks automatically associates the appropriate built-in security group with each layer (default setting).

  You can associate additional security groups with a layer after you create it but you cannot delete the built-in security group.
- **No** - OpsWorks Stacks does not associate built-in security groups with layers.

  You must create appropriate EC2 security groups and associate a security group with each layer that you create. However, you can still manually associate a built-in security group with a layer on creation; custom security groups are required only for those layers that need custom settings.

Note the following:

- If **Use OpsWorks security groups** is set to **Yes**, you cannot restrict a default security group's port access settings by adding a more restrictive security group to a layer. With multiple security groups, Amazon EC2 uses the most permissive settings. In addition, you cannot create more restrictive settings by modifying the built-in security group configuration. When you create a stack, OpsWorks Stacks overwrites the built-in security groups' configurations with the standard settings, so any changes that you make will be lost the next time you create a stack. If a layer requires more restrictive security group settings than the built-in security group, set **Use**

OpsWorks security groups to **No**, create custom security groups with your preferred settings, and assign them to the layers on creation.

- If you accidentally delete an OpsWorks Stacks security group and want to recreate it, it must an exact duplicate of the original, including the group name's capitalization. Instead of recreating the group manually, we recommend having OpsWorks Stacks perform the task for you. Just create a new stack in the same AWS region—and VPC, if present—and OpsWorks Stacks will automatically recreate all the built-in security groups, including the one that you deleted. You can then delete the stack if you don't have any further use for it; the security groups will remain.

Running a Stack in a VPC

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can control user access to a stack's instances by creating it in a virtual private cloud (VPC). For example, you might not want users to have direct access to your stack's app servers or databases and instead require that all public traffic be channeled through an elastic load balancer.

The basic procedure for running a stack in a VPC is:

1. Create an appropriately configured VPC, by using the Amazon VPC console or API, or an AWS CloudFormation template.
2. Specify the VPC ID when you create the stack.
3. Launch the stack's instances in the appropriate subnet.

The following briefly describes how VPCs work in OpsWorks Stacks.

> **Important**
>
> If you use the VPC Endpoint feature, be aware that that each instance in the stack must be able to complete the following actions from Amazon Simple Storage Service (Amazon S3):

- Install the instance agent.
- Install assets, such as Ruby.
- Upload Chef run logs.
- Retrieve stack commands.

To enable these actions, you must ensure that the stack's instances have access to the following buckets that match the stack's region. Otherwise, the preceding actions will fail. For Chef 12 Linux and Chef 12.2 Windows, the buckets are as follows.

Agent Buckets	Asset Buckets	Log Buckets	DNA Buckets
• opsworks-instance-agent-sa-east-1 • opsworks-instance-agent-ap-south-1 • opsworks-instance-agent-ap-northeast-1 • opsworks-instance-agent-ap-northeast-2 • opsworks-instance-agent-ap-southeast-1 • opsworks-instance-agent-ap-southeast-2 • opsworks-instance-agent-ca-central-1 • opsworks-instance-agent-eu-central-1 • opsworks-instance-agent-eu-west-1 • opsworks-instance-agent-eu-west-2	• opsworks-instance-assets-us-east-2 • opsworks-instance-assets-us-east-1 • opsworks-instance-assets-ap-south-1 • opsworks-instance-assets-ap-northeast-1 • opsworks-instance-assets-ap-northeast-2 • opsworks-instance-assets-ap-southeast-1 • opsworks-instance-assets-ap-southeast-2 • opsworks-instance-assets-ca-central-1 • opsworks-instance-assets-eu-central-1 • opsworks-instance-assets-eu-west-1	• opsworks-us-east-2-log • opsworks-us-east-1-log • opsworks-ap-south-1-log • opsworks-ap-northeast-1-log • opsworks-ap-northeast-2-log • opsworks-ap-southeast-1-log • opsworks-ap-southeast-2-log • opsworks-ca-central-1-log • opsworks-eu-central-1-log • opsworks-eu-west-1-log • opsworks-eu-west-2-log • opsworks-eu-west-3-log • opsworks-sa-east-1-log • opsworks-us-west-1-log • opsworks-us-west-2-log	• opsworks-us-east-2-dna • opsworks-us-east-1-dna • opsworks-ap-south-1-dna • opsworks-ap-northeast-1-dna • opsworks-ap-northeast-2-dna • opsworks-ap-southeast-1-dna • opsworks-ap-southeast-2-dna • opsworks-ca-central-1-dna • opsworks-eu-central-1-dna • opsworks-eu-west-1-dna • opsworks-eu-west-2-dna • opsworks-eu-west-3-dna • opsworks-sa-east-1-dna • opsworks-us-west-1-dna • opsworks-us-west-2-dna

Agent Buckets	Asset Buckets	Log Buckets	DNA Buckets
• opsworks-instance-agent-eu-west-3 • opsworks-instance-agent-us-east-1 • opsworks-instance-agent-us-east-2 • opsworks-instance-agent-us-west-1 • opsworks-instance-agent-us-west-2	• opsworks-instance-assets-eu-west-2 • opsworks-instance-assets-eu-west-3 • opsworks-instance-assets-sa-east-1 • opsworks-instance-assets-us-west-1 • opsworks-instance-assets-us-west-2		

For Chef 11.10 and earlier versions for Linux, the buckets are as follows. Chef 11.4 stacks are not supported in regional endpoints outside the US East (N. Virginia) Region.

Agent Buckets	Asset Buckets	Log Buckets	DNA Buckets
• opsworks-instance-agent-us-east-2 • opsworks-instance-agent-us-east-1 • opsworks-instance-agent-ap-south-1 • opsworks-instance-agent-ap-northeast-1 • opsworks-instance-agent-ap-northeast-2 • opsworks-instance-agent-ap-southeast-1 • opsworks-instance-agent-ap-southeast-2 • opsworks-instance-agent-ca-central-1 • opsworks-instance-agent-eu-central-1 • opsworks-instance-agent-eu-west-1	• opsworks-instance-assets-us-east-2 • opsworks-instance-assets-us-east-1 • opsworks-instance-assets-ap-south-1 • opsworks-instance-assets-ap-northeast-1 • opsworks-instance-assets-ap-northeast-2 • opsworks-instance-assets-ap-southeast-1 • opsworks-instance-assets-ap-southeast-2 • opsworks-instance-assets-ca-central-1 • opsworks-instance-assets-eu-central-1 • opsworks-instance-assets-eu-west-1	• prod_stage-log	• prod_stage-dna

Agent Buckets	Asset Buckets	Log Buckets	DNA Buckets
• opsworks-instance-agent-eu-west-2 • opsworks-instance-agent-eu-west-3 • opsworks-instance-agent-us-east-1 • opsworks-instance-agent-us-west-1 • opsworks-instance-agent-us-west-2	• opsworks-instance-assets-eu-west-2 • opsworks-instance-assets-eu-west-3 • opsworks-instance-assets-sa-east-1 • opsworks-instance-assets-us-west-1 • opsworks-instance-assets-us-west-2		

For more information, see VPC Endpoints.

Note

For OpsWorks Stacks to connect to the VPC endpoints that you enable, you must also configure routing for your NAT or public IP, as the OpsWorks Stacks agent still requires access to the public endpoint.

Topics

- VPC Basics
- Create a VPC for an OpsWorks Stacks Stack

VPC Basics

For a detailed discussion of VPCs, see Amazon Virtual Private Cloud. Briefly, a VPC consists of one or more *subnets*, each of which contains one or more instances. Each subnet has an associated routing table that directs outbound traffic based on its destination IP address.

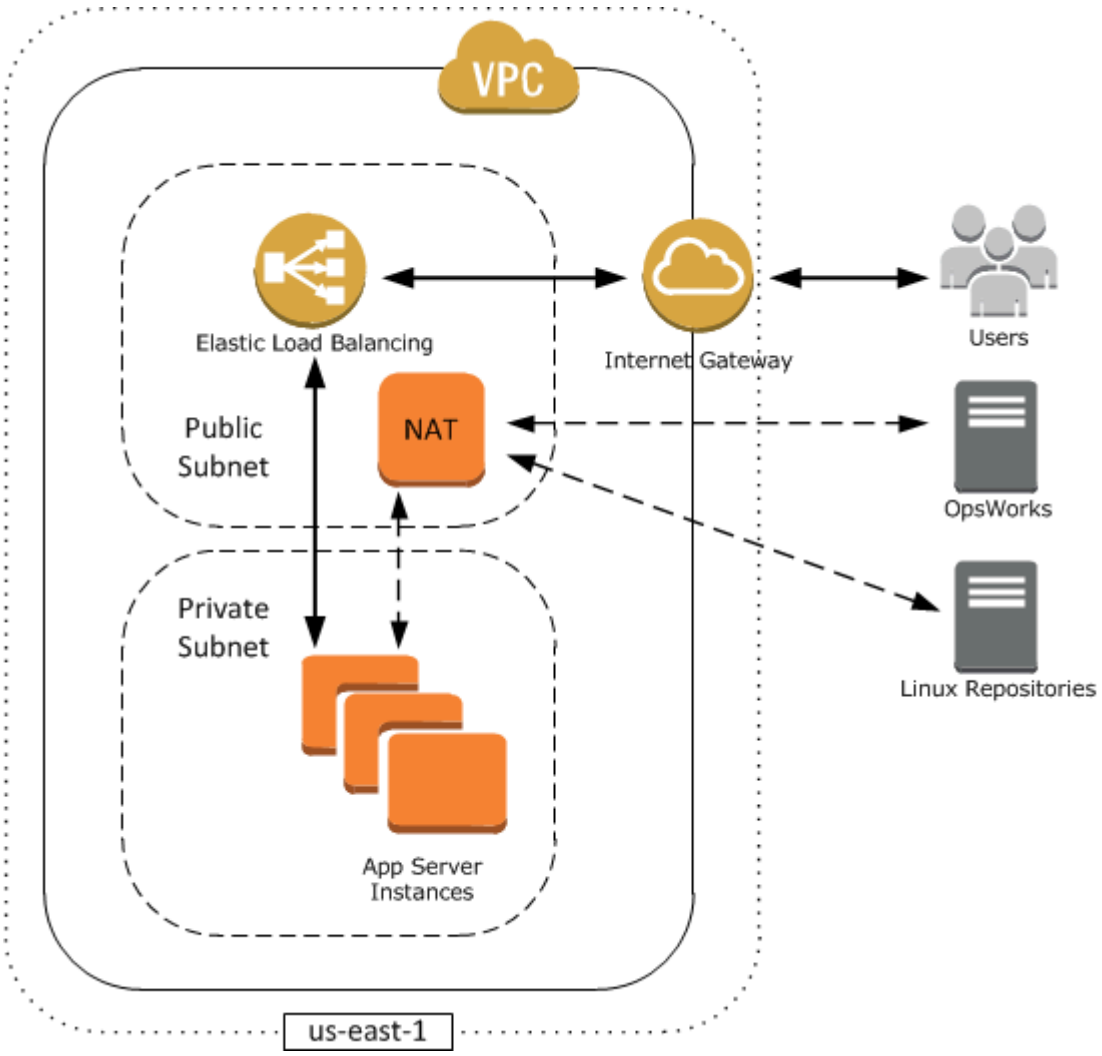
- Instances within a VPC can communicate with each other by default, regardless of their subnet. However, changes to network access control lists (ACLs), security group policies, or using static IP addresses can break this communication.
- Subnets whose instances can communicate with the Internet are referred to as *public subnets*.
- Subnets whose instances can communicate only with other instances in the VPC and cannot communicate directly with the Internet are referred to as *private subnets*.

OpsWorks Stacks requires the VPC to be configured so that every instance in the stack, including instances in private subnets, has access to the following endpoints:

- One of the OpsWorks Stacks service endpoints listed in the "Region Support" section of Getting Started with OpsWorks Stacks.
- One of the following instance service endpoints, used by the OpsWorks Stacks agent. The agent runs on managed customer instances to exchange data with the service.
  - opsworks-instance-service.us-east-2.amazonaws.com
  - opsworks-instance-service.us-east-1.amazonaws.com
  - opsworks-instance-service.us-west-1.amazonaws.com
  - opsworks-instance-service.us-west-2.amazonaws.com
  - opsworks-instance-service.ap-south-1.amazonaws.com
  - opsworks-instance-service.ap-northeast-1.amazonaws.com
  - opsworks-instance-service.ap-northeast-2.amazonaws.com
  - opsworks-instance-service.ap-southeast-1.amazonaws.com
  - opsworks-instance-service.ap-southeast-2.amazonaws.com
  - opsworks-instance-service.ca-central-1.amazonaws.com
  - opsworks-instance-service.eu-central-1.amazonaws.com
  - opsworks-instance-service.eu-west-1.amazonaws.com
  - opsworks-instance-service.eu-west-2.amazonaws.com
  - opsworks-instance-service.eu-west-3.amazonaws.com

- Amazon S3
- Any package repositories that your operating system depends on, such as the Amazon Linux or Ubuntu Linux repositories.
- Your app and custom cookbook repositories.

There are a variety of ways to configure a VPC to provide this connectivity. The following is a simple example of how you could configure a VPC for an OpsWorks Stacks app server stack.



This VPC has several components:

Subnets

The VPC has two subnets, one public and one private.

- The public subnet contains a load balancer and a network address translation (NAT) device, which can communicate with external addresses and with the instances in the private subnet.

- The private subnet contains the application servers, which can communicate with the NAT and load balancer in the public subnet but cannot communicate directly with external addresses.

Internet gateway

The Internet gateway allows instances with public IP addresses, such as the load balancer, to communicate with addresses outside the VPC.

Load balancer

The Elastic Load Balancing load balancer takes incoming traffic from users, distributes it to the app servers in the private subnet, and returns the responses to users.

NAT

The (NAT) device provides the app servers with limited Internet access, which is typically used for purposes such as downloading software updates from an external repository. All OpsWorks Stacks instances must be able to communicate with OpsWorks Stacks and with the appropriate Linux repositories. One way to handle this issue is to put a NAT device with an associated Elastic IP address in a public subnet. You can then route outbound traffic from instances in the private subnet through the NAT.

> **Note**
>
> A single NAT instance creates a single point of failure in your private subnet's outbound traffic. You can improve reliability by configuring the VPC with a pair of NAT instances that take over for each other if one fails. For more information, see High Availability for Amazon VPC NAT Instances. You can also use a NAT gateway. For more information, see NAT in the Amazon VPC User Guide.

The optimal VPC configuration depends on your OpsWorks Stacks stack. The following are a few examples of when you might use certain VPC configurations. For examples of other VPC scenarios, see Scenarios for Using Amazon VPC.

Working with one instance in a public subnet

If you have a single-instance stack with no associated private resources—such as an Amazon RDS instance that should not be publicly accessible—you can create a VPC with one public subnet and put the instance in that subnet. If you are not using a default VPC, you must have

the instance's layer assign an Elastic IP address to the instance. For more information, see OpsWorks Layer Basics.

Working with private resources

If you have resources that should not be publicly accessible, you can create a VPC with one public subnet and one private subnet. For example, in a load-balanced automatic scaling environment, you can put all the Amazon EC2 instances in the private subnet and the load balancer in a public subnet. That way the Amazon EC2 instances cannot be directly accessed from the Internet; all incoming traffic must be routed through the load balancer.

The private subnet isolates the instances from Amazon EC2 direct user access, but they must still send outbound requests to AWS and the appropriate Linux package repositories. To allow such requests you can, for example, use a network address translation (NAT) device with its own Elastic IP address and then route the instances' outbound traffic through the NAT. You can put the NAT in the same public subnet as the load balancer, as shown in the preceding example.

- If you are using a back-end database such as an Amazon RDS instance, you can put those instances in the private subnet. For Amazon RDS instances, you must specify at least two different subnets in different Availability Zones.
- If you require direct access to instances in a private subnet—for example, you want to use SSH to log in to an instance—you can put a bastion host in the public subnet that proxies requests from the Internet.

Extending your own network into AWS

If you want to extend your own network into the cloud and also directly access the Internet from your VPC, you can create a VPN gateway. For more information, see Scenario 3: VPC with Public and Private Subnets and Hardware VPN Access.

Create a VPC for an OpsWorks Stacks Stack

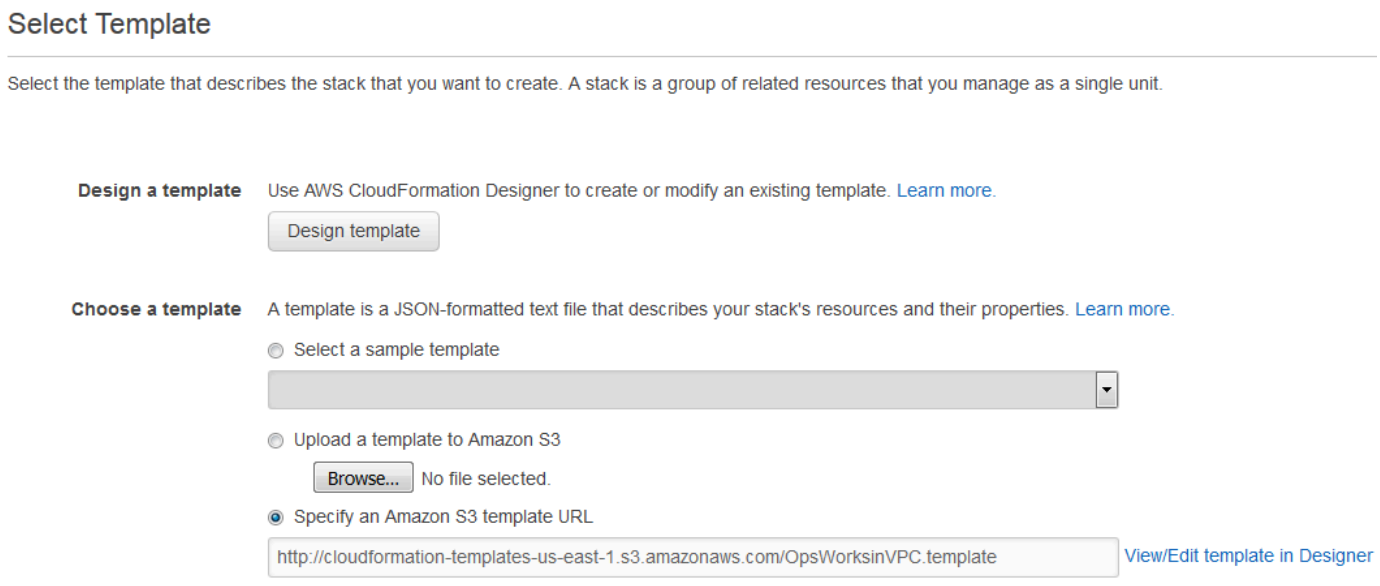
This section shows how to create a VPC for an OpsWorks Stacks stack by using an example AWS CloudFormation template. You can download the template in the OpsWorksVPCtemplates.zip file. For more information on how to manually create a VPC like the one discussed in this topic, see Scenario 2: VPC with Public and Private Subnets. For details on how to configure routing tables, security groups, and so on, see the example template.

> **Note**
>
> By default, OpsWorks Stacks displays subnet names by concatenating their CIDR range and Availability Zone, such as `10.0.0.1/24 - us-east-1b`. To make the names more readable, create a tag for each subnet with **Key** set to **`Name`** and **Value** set to the subnet name. OpsWorks Stacks then appends the subnet name to the default name. For example, the private subnet in the following example has a tag with **Name** set to **`Private`**, which OpsWorks displays as `10.0.0.1/24 us-east - 1b - Private`.

You can launch a VPC template using the AWS CloudFormation console with just a few steps. The following procedure uses the example template to create a VPC in US East (N. Virginia) Region. For directions on how to use the template to create a VPC in other regions, see the note that follows the procedure.

To create the VPC

1. Open the AWS CloudFormation console, select the **US East (N. Virginia)** region, and choose **Create Stack**.
2. On the **Select Template** page, select **Upload a template**. Browse for the `OpsWorksinVPC.template` file that you downloaded in the OpsWorksVPCtemplates.zip file.Choose **Continue**.



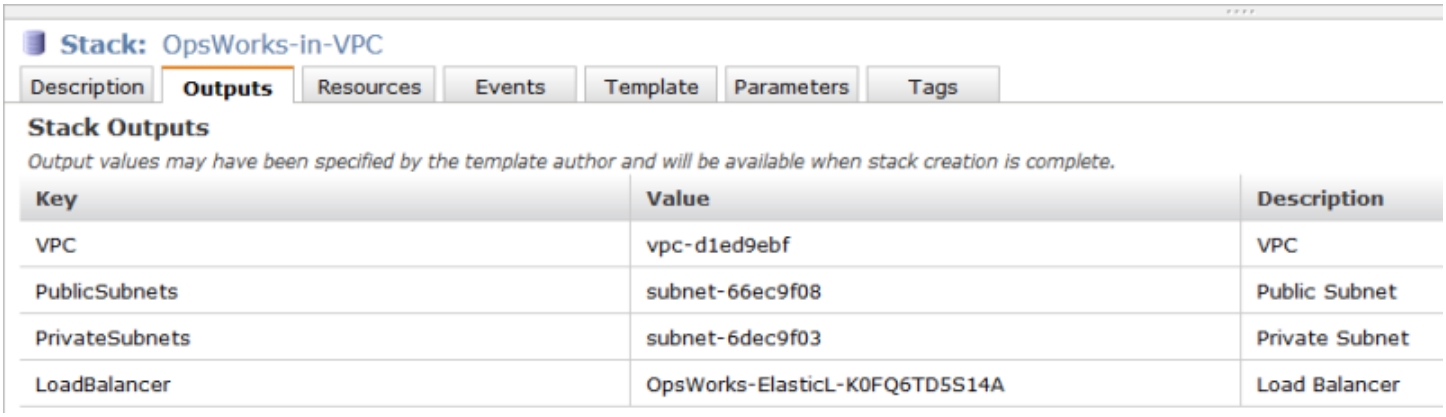
You can also launch this stack by opening AWS CloudFormation Sample Templates, locating the OpsWorks Stacks VPC template, and choosing **Launch Stack**.

3. On the **Specify Parameters** page, accept the default values and choose **Continue**.
4. On the **Add Tags** page, create a tag with **Key** set to **Name** and **Value** set to the VPC name. This tag will make it easier to identify your VPC when you create an OpsWorks Stacks stack.
5. Choose **Continue** and then **Close** to launch the stack.

Note: You can create the VPC in other regions by using either of the following approaches.

- Go to Using Templates in Different Regions, choose the appropriate region, locate the OpsWorks Stacks VPC template, and then choose **Launch Stack**.
- Copy the template file to your system, select the appropriate region in the AWS CloudFormation console, and use the **Create Stack** wizard's **Upload a template to Amazon S3** option to upload the template from your system.

The example template includes outputs that provide the VPC, subnet, and load balancer IDs that you will need to create the OpsWorks Stacks stack. You can see them by choosing the **Outputs** tab at the bottom of the AWS CloudFormation console window.



Update a Stack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

After you have created a stack, you can update the configuration at any time. On the **Stack** page, click **Stack Settings**, and then click **Edit**, which displays the **Settings** page. Make the changes that you want and click **Save**.

The settings are the same as those discussed in Create a New Stack. Refer to that topic for details. However, note the following:

- You cannot modify the region or VPC ID.
- If your stack is running in a VPC, the settings include a **Default subnet** setting, which lists the VPC's subnets. If your stack is not running in a VPC, the setting is labeled **Default Availability Zones**, and lists the region's Availability Zones.
- You can change the default operating system, but you cannot specify a Linux operating system for a Windows stack, or Windows for a Linux stack.
- If you change any of the default instance settings, such as **Hostname theme** or **Default SSH key**, the new values apply only to any new instances you create, not to existing instances.
- Changing the **Name** changes the name that is displayed by the console; it does not change the underlying short name that OpsWorks Stacks uses to identify the stack.
- Before you change **Use OpsWorks security groups** from **Yes** to **No**, each layer must have at least one security group in addition to the layer's built-in security group. For more information, see Editing an OpsWorks Layer's Configuration.

  OpsWorks Stacks then deletes the built-in security groups from every layer.
- If you change **Use OpsWorks security groups** from **No** to **Yes**, OpsWorks Stacks adds the appropriate built-in security group to each layer but does not delete the existing security groups.

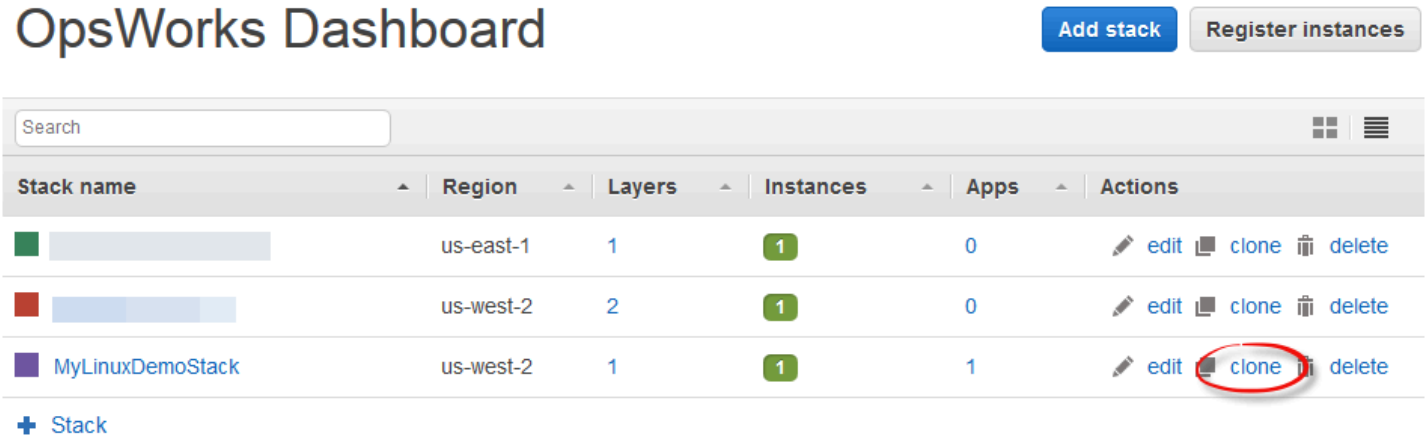
Clone a Stack

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

It is sometimes useful to create multiple copies of a stack. For example, you might want to add redundancy as a disaster recovery or prevention measure, or you might use an existing stack as a starting point for a new stack. The simplest approach is to clone the source stack. On the OpsWorks Stacks dashboard, in the **Actions** column of the row for the stack that you want to clone, choose **clone**, which opens the **Clone stack** page.



Initially, the settings for the cloned stack are identical to those for the source stack except that the word "copy" is appended to the stack name. For information about these settings, see Create a New Stack. There are also two additional, optional settings:

Permissions

If **all permissions** is selected (the default), the source stack permissions are applied to the cloned stack.

Apps

Lists apps that are deployed on the source stack. For each app listed, if the corresponding check box is selected (the default), the app is deployed to the cloned stack.

> **Note**
>
> You cannot clone a stack from one regional endpoint to another; for example, you cannot clone a stack from the US West (Oregon) Region (us-west-2) to the Asia Pacific (Mumbai) Region (ap-south-1).

When you have finalized the settings, choose **Clone stack**. OpsWorks Stacks creates a new stack that consists of the source stack's layers and optionally its apps and permissions. The layers have the same configuration as the originals, subject to any modifications that you made. However, cloning does not create any instances. You must add an appropriate set of instances to each layer of the cloned stack and then start them. As with any stack, you can perform normal management tasks on a cloned stack, such as adding, deleting, or modifying layers or adding and deploying apps.

To make the cloned stack operational, start the instances. OpsWorks Stacks sets up and configures each instance according to its layer membership. It also deploys any applications, just as it does with a new stack.

Run OpsWorks Stacks Stack Commands

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks provides a set of *stack commands*, which you can use to perform a variety of operations on a stack's instances. To run a stack command, click **Run Command** on the **Stack** page. You then choose the appropriate command, specify any options, and press the button at the lower right, which will be labeled with the command's name.

> **Note**
>
> OpsWorks Stacks also supports a set of *deployment commands*, which you use to manage app deployment. For more information, see Deploying Apps.

You can run the following stack commands on any stack.

Update Custom Cookbooks

Updates the instances' custom cookbooks with the current version from the repository. This command does not run any recipes. To run the updated recipes, you can use an `Execute Recipes`, `Setup`, or `Configure` stack command, or redeploy your application to run the Deploy recipes. For more information on custom cookbooks, see Cookbooks and Recipes.

Execute Recipes

Executes a specified set of recipes on the instances. For more information, see Manually Running Recipes.

Setup

Runs the instances' Setup recipes.

Configure

Runs the instances' Configure recipes.

> **Note**
>
> To use **Setup** or **Configure** to run recipes on an instance, the recipes must be assigned to the corresponding lifecycle event for the instance's layer. For more information, see Executing Recipes.

You can run the following stack commands only on Linux-based stacks.

Install Dependencies

Installs the instances' packages. Starting in Chef 12, this command is not available.

Update Dependencies

(Linux only. Starting in Chef 12, this command is not available.) Installs regular operating system updates and package updates. The details depend on the instances' operating system. For more information, see Managing Security Updates.

Use the **Upgrade Operating System** command to upgrade instances to a new Amazon Linux version.

Upgrade Operating System

(Linux only) Upgrades the instances' Amazon Linux operating systems to the latest version. For more information, see OpsWorks Stacks operating systems.

> **Important**
>
> After running **Upgrade Operating System**, we recommend that you also run **Setup**. This ensures that services are correctly restarted.

Stack commands have the following options, some of which appear only for certain commands.

Comment

(Optional) Enter any custom remarks you care to add.

Recipes to execute

(Required) This setting appears only if you select the **Execute Recipes** command. Enter the recipes to be executed using the standard *`cookbook_name`*::*`recipe_name`* format, separated by commas. If you specify multiple recipes, OpsWorks Stacks executes them in the listed order.

Allow reboot

(Optional) This setting appears only if you select the **Upgrade Operating System** command. The default value is **Yes**, which directs OpsWorks Stacks to reboot the instances after installing the upgrade.

Custom Chef JSON

(Optional) Choose **Advanced** to display this option, which allows you to specify custom JSON attributes to be incorporated into the stack configuration and deployment attributes.

Instances

(Optional) Specify the instances on which to execute the command. All online instances are selected by default. To run the command on a subset of instances, select the appropriate layers or instances.

> **Note**
>
> You might see execute_recipes executions that you did not run listed on the **Deployment** and **Commands** pages. This is usually the result of a permissions change, such as granting or removing SSH permissions for a user. When you make such a change, OpsWorks Stacks uses execute_recipes to update permissions on the instances.

Using Custom JSON

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Several OpsWorks Stacks actions allow you to specify custom JSON, which OpsWorks Stacks installs on instances and can be used by recipes.

You can specify custom JSON in the following situations:

- When you create, update, or clone a stack.

  AWS OpsWorks Stacks installs the custom JSON on all instances for all subsequent lifecycle events.
- When you run a deployment or stack command.

  AWS OpsWorks Stacks passes the custom JSON to instances only for that event.

Custom JSON must be represented by, and formatted as, a valid JSON object. For example:

```
{
  "att1": "value1",
  "att2": "value2"
  ...
}
```

OpsWorks Stacks stores custom JSON in the following locations:

On Linux instances:

- `/var/chef/runs/`*`run-ID`*`/attribs.json`
- `/var/chef/runs/`*`run-ID`*`/nodes/`*`hostname`*`.json`

On Windows instances:

- *`drive`*`:\chef\runs\`*`run-ID`*`\attribs.json`
- *`drive`*`:\chef\runs\`*`run-ID`*`\nodes\`*`hostname`*`.json`

> **Note**
>
> In Chef 11.10 and earlier versions for Linux, the custom JSON is located in the following path on Linux instances, Windows instances are not available, and there is no `attribs.json` file. The logs are stored in the same folder or directory as the JSON. For more information about custom JSON in Chef 11.10 and earlier versions for Linux, see Overriding Attributes with Custom JSON and Chef Logs.
> `/var/lib/aws/opsworks/chef/`*`hostname`*`.json`

In the preceding paths, *`run-ID`* is a unique ID that OpsWorks Stacks assigns to each Chef run on an instance, and *`hostname`* is the instance's hostname.

To access custom JSON from Chef recipes, use standard Chef node syntax.

For example, suppose that you want to define simple settings for an app that you want to deploy, such as whether the app is initially visible and the app's initial foreground and background colors. Suppose you define these app settings with a JSON object as follows:

```
{
  "state": "visible",
  "colors": {
    "foreground": "light-blue",
    "background": "dark-gray"
  }
}
```

To declare the custom JSON for a stack:

1. On the stack page, choose **Stack Settings**, and then choose **Edit**.
2. For **Custom Chef JSON**, type the JSON object, and then choose **Save**.

> **Note**
>
> You can declare custom JSON at the deployment, layer, and stack levels. You may want to do this if you want some custom JSON to be visible only to an individual deployment or layer. Or, for example, you may want to temporarily override custom JSON declared at the stack level with custom JSON declared at the layer level. If you declare custom JSON at multiple levels, custom JSON declared at the deployment level overrides any custom JSON declared at both the layer and stack levels. Custom JSON declared at the layer level overrides any custom JSON declared only at the stack level.
>
> To use the OpsWorks Stacks console to specify custom JSON for a deployment, on the **Deploy App** page, choose **Advanced**. Type the custom JSON in the **Custom Chef JSON** box, and then choose **Save**.
>
> To use the OpsWorks Stacks console to specify custom JSON for a layer, on the **Layers** page, choose **Settings** for the desired layer. Type the custom JSON in the **Custom JSON** box, and then choose **Save**.
>
> For more information, see Editing an OpsWorks Layer's Configuration and Deploying Apps.

When you run a deployment or stack command, recipes can retrieve these custom values by using standard Chef node syntax, which maps directly to the hierarchy in the custom JSON object. For example, the following recipe code writes messages to the Chef log about the preceding custom JSON values:

```
Chef::Log.info("********** The app's initial state is '#{node['state']}' **********")
```

```
Chef::Log.info("********** The app's initial foreground color is '#{node['colors']
['foreground']}' **********")
Chef::Log.info("********** The app's initial background color is '#{node['colors']
['background']}' **********")
```

This approach can be useful for passing data to recipes. OpsWorks Stacks adds that data to the instance, and recipes can retrieve the data by using standard Chef node syntax.

> **Note**
>
> Custom JSON is limited to 120 KB. If you need more capacity, we recommend storing some of the data on Amazon Simple Storage Service (Amazon S3). Your custom recipes can then use the AWS CLI or the AWS SDK for Ruby to download the data from the Amazon S3 bucket to your instance.

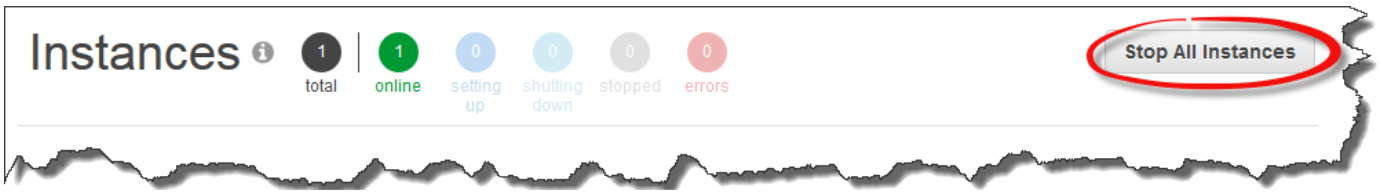
Delete a Stack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

If you no longer need a stack, you can delete it. Only empty stacks can be deleted; you must first delete all instances, apps, and layers in the stack.

To delete a stack

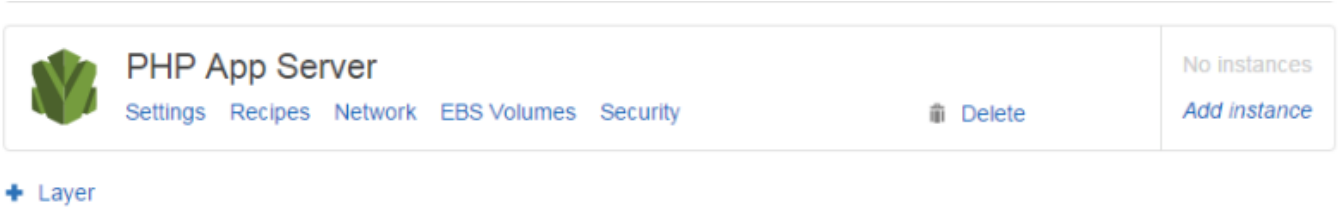
1. On the OpsWorks Stacks dashboard, choose the stack that you want to shut down and delete.
2. In the navigation pane, choose **Instances**.
3. On the **Instances** page, choose **Stop all Instances**.



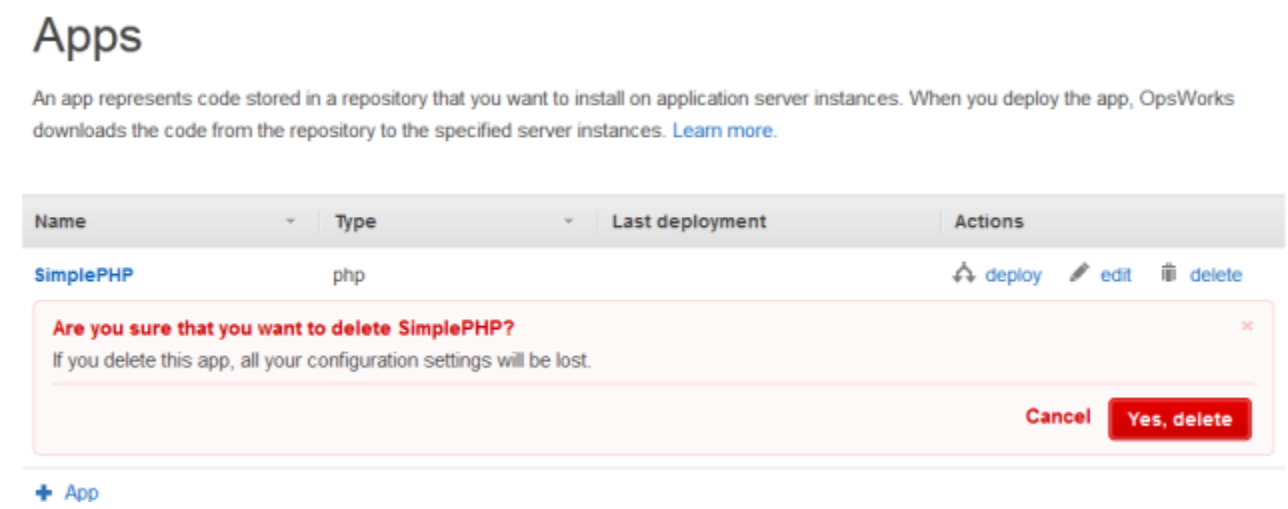
4. After the instances have stopped, for each instance in the layer, choose **delete** in the **Actions** column. When prompted to confirm, choose **Yes, Delete**.



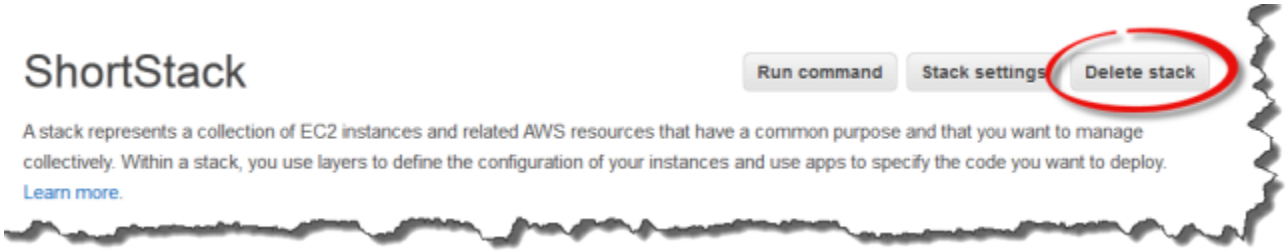
5. When all the instances are deleted, in the navigation pane, choose **Layers**.
6. On the **Layers** page, for each layer in the stack, choose **delete**. On the confirmation prompt, choose **Yes, Delete**.



7. When all layers are deleted, in the navigation pane, choose **Apps**.
8. On the **Apps** page, for each app in the stack, choose **delete** in the **Actions** column. On the confirmation prompt, choose **Yes, Delete**.



9. When all apps are deleted, in the navigation pane, choose **Stack**.
10. On the stack page, choose **Delete stack**. On the confirmation prompt, choose **Yes, Delete**.



Deleting Other AWS Resources Used by a Stack

You use other AWS resources with OpsWorks Stacks to create and manage your stacks. As you delete a stack, consider also deleting resources that worked with with the stack, if another stack is not using them, and resources outside OpsWorks Stacks are not using them. The following are suggested reasons for cleaning up external AWS resources that you used with a stack.

- External AWS resources can continue to accrue charges on your AWS account.
- Resources such as Amazon S3 buckets can contain personally-identifying, sensitive, or confidential information.

> **Important**
>
> Do not delete these resources if they are in use by other stacks. Note that IAM roles and security groups are global, so stacks in other regions might be using these same resources.

The following are other AWS resources that stacks use, and links to information about how to delete them.

Service roles and instance profiles

When you create a stack, you specify an IAM role and an instance profile that OpsWorks Stacks uses to create allowed resources on your behalf. OpsWorks creates the role and instance profile for you if you do not choose existing ones. The role and instance profile that OpsWorks creates for you are named `aws-opsworks-service-role` and `aws-opsworks-ec2-role`, respectively. If no other stacks in your account are using the IAM role and instance profile, it is safe to delete these resources. For information about how to delete IAM roles and instance profiles, see Deleting roles or instance profiles in the *IAM User Guide*.

Security groups

In OpsWorks Stacks, you can specify user-defined security groups at the layer level. You create security groups by using the Amazon EC2 console or API. Stacks and layers in other regions can use the same security groups, because security groups are global. You can delete a security group if it's not in use by other AWS resources. For more information about how to delete a security group, see Delete a security group in the *Amazon EC2 User Guide*.

Amazon EBS volumes

In OpsWorks Stacks, you add EBS volumes at the layer level, and they are attached to instances in the layer. You create EBS volumes by using the Amazon EC2 service console or API, then attach them to OpsWorks Stacks instances at the layer level. EBS volumes are specific to an availability zone. If you are no longer using an EBS volume in any stacks in a specific region and availability zone, you can delete the volume. For more information about how to delete an Amazon EBS volume, see Deleting an Amazon EBS Volume in the *Amazon EC2 User Guide*.

Amazon Simple Storage Service (Amazon S3) buckets

In OpsWorks Stacks, you can use Amazon S3 buckets for the following. Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

- Storing app code
- Storing cookbooks and recipes
- CloudTrail logs, if you have enabled CloudTrail logging in OpsWorks Stacks
- Amazon CloudWatch Logs streams, if you have enabled them in OpsWorks Stacks

Elastic IP addresses

If you registered Elastic IP addresses with OpsWorks Stacks, and you no longer need the Elastic IP addresses, you can release the Elastic IP address.

Elastic Load Balancing load balancers

If you no longer need an Elastic Load Balancing classic load balancer that you have been using with layers in your stack, you can delete it. For more information, see Delete Your Load Balancer in the *User Guide for Classic Load Balancers*.

Amazon Relational Database Service (Amazon RDS) instances

If you registered Amazon RDS database (DB) instances with OpsWorks Stacks, and you no longer need them, you can delete DB instances. For more information about how to delete DB instances, see Deleting a DB Instance in the *Amazon RDS User Guide*.

Amazon Elastic Container Service (Amazon ECS) clusters

If your stack included ECS cluster layers, and you are no longer using the ECS cluster that was registered with a layer, you can delete the ECS cluster. For more information about how to delete an ECS cluster, see Deleting a Cluster in the *Amazon ECS Developer Guide*.

Layers

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Every stack contains one or more layers, each of which represents a stack component, such as a load balancer or a set of application servers.

As you work with OpsWorks Stacks layers, keep the following in mind:

- Each layer in a stack must have at least one instance and can optionally have multiple instances.

- Each instance in a stack must be a member of at least one layer, except for registered instances.

  You cannot configure an instance directly, except for some basic settings such as the SSH key and hostname. You must create and configure an appropriate layer, and add the instance to the layer.

Amazon EC2 instances can optionally be a member of multiple layers. In that case, OpsWorks Stacks runs the recipes to install and configure packages, deploy applications, and so on for each of the instance's layers.

By assigning an instance to multiple layers, you could, for example do the following:

- Reduce expenses by hosting the database server and the load balancer on a single instance.
- Use one of your application servers for administration.

  Create a custom administrative layer and add one of the application server instances to that layer. The administrative layer's recipes configure that application server instance to perform administrative tasks, and install any additional required software. The other application server instances are just application servers.

This section describes how to work with layers.

Topics

- OpsWorks Layer Basics
- Elastic Load Balancing Layer
- Amazon RDS Service Layer
- ECS Cluster Layers
- Custom OpsWorks Stacks Layers
- Per-layer Operating System Package Installations

OpsWorks Layer Basics

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section describes how to perform operations that are common to all OpsWorks Stacks layers.

Topics

- Creating an OpsWorks Layer
- Editing an OpsWorks Layer's Configuration
- Using Auto Healing to Replace Failed Instances
- Deleting an OpsWorks Layer

Creating an OpsWorks Layer

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

When you create a new stack, you see the following page:



To add the first OpsWorks layer

1. Click **Add a Layer**.
2. On the **Add Layer** page, select the appropriate layer, which displays the layer's configuration options.
3. Configure the layer appropriately and click **Add Layer** to add it to the stack. The following sections describe how to configure the various layers.

   > **Note**
   >
   > The **Add Layer** page displays only the more commonly used configuration settings for each layer. You can specify additional settings by editing the layer.

4. Add instances to the layer and start them.

   > **Note**
   >
   > If an instance is a member of multiple layers, you must add it to all of them before you start the instance. You cannot add an online instance to a layer.

To add more layers, open the **Layers** page and click **+ Layer** to open the **Add Layer** page.

When you start an instance, OpsWorks Stacks automatically runs the Setup and Deploy recipes for each of the instance's layers to install and configure the appropriate packages and deploy the appropriate applications. You can customize a layer's setup and configuration process in a variety of ways, such as by assigning custom recipes to the appropriate lifecycle events. OpsWorks Stacks runs custom recipes after the standard recipes for each event. For more information, see Cookbooks and Recipes.

The following layer-specific sections describe how handle Steps 2 and 3 for the various OpsWorks Stacks layers. For more information how to add instances, see Adding an Instance to a Layer.

Editing an OpsWorks Layer's Configuration

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate

their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

After you create a layer, some properties (such as AWS region) are immutable, but you can change most of the layer configuration at any time. Editing the layer also provides access to configuration settings that are not available on the **Add Layer** page. The settings take effect as soon as you save the new configuration.

To edit an OpsWorks layer

1. In the navigation pane, click **Layers**.
2. On the **Layers** page, choose a layer name to open the details page, which shows the current configuration.

   > **Note**
   >
   > Choosing one of the names under the layer name takes you directly to the associated tab on the details page.

3. Click **Edit** and then select the appropriate tab: **General Settings**, **Recipes**, **Network**, **EBS Volumes**, or **Security**.

The following sections describe the settings on the various tabs that are available to all layers. Some layers have additional layer-specific settings, which appear at the top of the page. In addition, some settings are available only for Linux-based stacks, as noted.

Topics

- General Settings
- Recipes
- Network
- EBS Volumes
- Security
- CloudWatch Logs
- Tags

General Settings

All layers have the following settings:

Auto healing enabled

Whether auto healing is enabled for the layer's instances. The default setting is **Yes**.

Custom JSON

Data in JSON format that is passed to your Chef recipes for all instances in this layer. You can use this, for example, to pass data to your own recipes. For more information, see Using Custom JSON.

> **Note**
>
> You can declare custom JSON at the deployment, layer, and stack levels. You may want to do this if you want some custom JSON to be visible across the stack or only to an individual deployment. Or, for example, you may want to temporarily override custom JSON declared at the layer level with custom JSON declared at the deployment level. If you declare custom JSON at multiple levels, custom JSON declared at the deployment level overrides any custom JSON declared at both the layer and stack levels. Custom JSON declared at the layer level overrides any custom JSON declared only at the stack level.
>
> To use the OpsWorks Stacks console to specify custom JSON for a deployment, on the **Deploy App** page, choose **Advanced**. Type the custom JSON in the **Custom Chef JSON** box, and then choose **Save**.
>
> To use the OpsWorks Stacks console to specify custom JSON for a stack, on the stack settings page, type the custom JSON in the **Custom JSON** box, and then choose **Save**. For more information, see Using Custom JSON and Deploying Apps.

Instance shutdown timeout

Specifies how long (in seconds) OpsWorks Stacks waits after triggering a Shutdown lifecycle event before stopping or terminating the Amazon EC2 instance. The default setting is 120 seconds. The purpose of the setting is to give the instance's Shutdown recipes enough time to complete their tasks before terminating the instance. If your custom Shutdown recipes might require more time, modify the setting accordingly. For more information on instance shutdown, see Stopping an Instance.

The remaining settings on this tab vary with the type of layer and are identical to the settings on the layer's **Add Layer** page.

Recipes

The **Recipes** tab includes the following settings.

Custom Chef recipes

You can assign custom Chef recipes to the layer's lifecycle events. For more information, see Executing Recipes.

Network

The **Network** tab includes the following settings.

Elastic Load Balancing

You can attach an Elastic Load Balancing load balancer to any layer. OpsWorks Stacks then automatically registers the layer's online instances with the load balancer and deregisters them when they go offline. If you have enabled the load balancer's connection draining feature, you can specify whether OpsWorks Stacks supports it. For more information, see Elastic Load Balancing Layer.

Automatically Assign IP Addresses

You can control whether OpsWorks Stacks automatically assigns public or Elastic IP addresses to the layer's instances. Here's what happens when you enable this option:

- For instance store-backed instances, OpsWorks Stacks automatically assigns an address each time the instance is started.
- For Amazon EBS-backed instances, OpsWorks Stacks automatically assigns an address when the instance is started for the first time.
- If an instance belongs to more than one layer, OpsWorks Stacks automatically assigns an address if you have enabled automatic assignment for at least one of the layers,

> **Note**
>
> If you enable automatic assignment of public IP addresses, it applies only to new instances. OpsWorks Stacks cannot update the public IP address for existing instances.

If your stack is running in a VPC, you have separate settings for public and Elastic IP addresses. The following table explains how these interact:

Elastic IP addresses \ Public IP addresses	Yes	No
Yes	Instances receive an Elastic IP address when they are started for the first time, or a public IP address if an Elastic IP cannot be assigned.	Instances receive an Elastic IP address when they are started for the first time.
No	Instances receive a public IP address each time they are started.	Instances receive only a private IP address, which is not accessible from outside the VPC.

> **Note**
>
> Instances must have a way to communicate with the OpsWorks Stacks service, Linux package repositories, and cookbook repositories. If you specify no public or Elastic IP address, your VPC must include a component such as a NAT that allows the layer's instances to communicate with external sites. For more information, see Running a Stack in a VPC.

If your stack is not running in a VPC, **Elastic IP addresses** is your only setting:

- **Yes**: Instances receive an Elastic IP address when they are started for the first time, or a public IP address if an Elastic IP address cannot be assigned.
- **No**: Instances receive a public IP address each time they are started.

EBS Volumes

The **EBS Volumes** tab includes the following settings.

EBS optimized instances

Whether the layer's instances should be optimized for Amazon Elastic Block Store (Amazon EBS). For more information, see Amazon EBS-Optimized Instances.

Additional EBS Volumes

(Linux only) You can add Amazon EBS volumes to or remove them from the layer's instances. When you start an instance, OpsWorks Stacks automatically creates the volumes and attaches

them to the instances. You can use the **Resources** page to manage a stack's EBS volumes. For more information, see Resource Management.

- **Mount point** – (Required) Specify the mount point or directory where the EBS volume will be mounted.
- **# Disks** – (Optional) If you specified a RAID array, the number of disks in the array.

  Each RAID level has a default number of disks, but you can select a larger number from the list.
- **Size total (GiB)** – (Required) The volume's size, in GiB.

  For a RAID array, this setting specifies the total array size, not the size of each disk.

  The following table shows the minimum and maximum volume sizes allowed for each volume type.

Volume Type	Minimum Size (GiB)	Maximum Size (GiB)
Magnetic	1	1024
Provisioned IOPS (SSD)	4	16384
General Purpose (SSD)	1	16384
Throughput Optimized (HDD)	500	16384
Cold HDD	500	16384

- **Volume Type** – (Optional) Specify whether to create a magnetic, General Purpose SSD, Throughput Optimized HDD, Cold HDD, or PIOPS volume.

  The default value is **Magnetic**.
- **Encrypted**– (Optional) Specify whether to encrypt the contents of the EBS volume.
- **IOPS per disk** – (Required for Provisioned IOPS SSD and General Purpose SSD volumes) If you specify a Provisioned IOPS SSD or General Purpose SSD volume, you must also specify the **IOPS per disk**.

  For provisioned IOPS volumes, you can specify the IOPS rate when you create the volume. The ratio of IOPS provisioned and the volume size requested can be a maximum of 30 (in

other words, a volume with 3000 IOPS must be at least 100 GB). General Purpose (SSD) volume types have a baseline IOPS of volume size x 3 with a maximum of 10000 IOPS and can burst up to 3000 IOPS for 30 minutes.

When you add volumes to or remove them from a layer, note the following:

- If you add a volume, every new instance gets the new volume, but OpsWorks Stacks does not update the existing instances.
- If you remove a volume, it applies only to new instances; the existing instances retain their volumes.

Specifying a Mount Point

You can specify any mount point that you prefer. However, be aware that some mount points are reserved for use by OpsWorks Stacks or Amazon EC2 and should not be used for Amazon EBS volumes. Do not use typical Linux system folders such as `/home` or `/etc`.

The following mount points are reserved for use by OpsWorks Stacks.

- `/srv/www`
- `/var/log/apache2` (Ubuntu)
- `/var/log/httpd` (Amazon Linux)
- `/var/log/mysql`
- `/var/www`

When an instance boots or reboots, autofs (an automounting daemon) uses ephemeral device mount points such as `/media/ephemeral0` for bind mounts. This operation takes place before Amazon EBS volumes are mounted. To ensure that your Amazon EBS volume's mount point does not conflict with autofs, do not specify an ephemeral device mount point. The possible ephemeral device mount points depend on the particular instance type, and whether it is instance store–backed or Amazon EBS–backed. To avoid a conflict with autofs, do the following:

- Verify the ephemeral device mount points for the particular instance type and backing store that you want to use.
- Be aware that a mount point that works for an instance store–backed instance might conflict with autofs if you switch to an Amazon EBS–backed instance, or vice versa.

> **Note**
>
> If you want to change the instance store block device mapping, you can create a custom AMI. For more information, see Amazon EC2 Instance Store. For more information about how to create a custom AMI for OpsWorks Stacks, see Using Custom AMIs.

The following is an example of how to use a custom recipe to ensure that a volume's mount point doesn't conflict with autofs. You can adapt it as needed for your particular use case.

To avoid a conflicting mount point

1. Assign an Amazon EBS volume to desired layer but use a mount point such as `/mnt/workspace` that will never conflict with autofs.
2. Implement the following custom recipe, which creates an application directory on the Amazon EBS volume and links to it from `/srv/www/`. For more information on how to implement custom recipes, see Cookbooks and Recipes and Customizing OpsWorks Stacks.

```
mount_point = node['ebs']['raids']['/dev/md0']['mount_point'] rescue nil

if mount_point
  node[:deploy].each do |application, deploy|
    directory "#{mount_point}/#{application}" do
      owner deploy[:user]
      group deploy[:group]
      mode 0770
      recursive true
    end

    link "/srv/www/#{application}" do
      to "#{mount_point}/#{application}"
    end
  end
end
```

3. Add a `depends 'deploy'` line to the custom cookbook's `metadata.rb` file.
4. Assign this recipe to the layer's Setup event.

Security

The **Security** tab includes the following settings.

Security Groups

A layer must have at least one associated security group. You specify how to associate security groups when you create or update a stack. OpsWorks Stacks provides a standard set of built-in security groups.

- The default option is to have OpsWorks Stacks automatically associate the appropriate built-in security group with each layer.
- You can also choose to not automatically associate built-in security groups and instead associate a custom security group with each layer when you create the layer.

For more information on security groups, see Using Security Groups.

After the layer has been created, you can use **Security Groups** to add more security groups to the layer by selecting them from the **Custom security groups** list. After you add a security group to a layer, OpsWorks Stacks adds it to all new instances. (Note that instance-store instances that are restarted will be brought up as new instances, so they will also have the new security groups.) OpsWorks Stacks does not add security groups to online instances.

You can delete existing security groups by clicking the **x**, as follows:

- If you chose to have OpsWorks Stacks automatically associate built-in security groups, you can delete custom security groups that you added earlier by clicking the **x**, but you cannot delete the built-in group.
- If you chose to not automatically associate built-in security groups, you can delete any existing security groups, including the original one, as long as the layer retains at least one group.

After you remove a security group from a layer, OpsWorks Stacks does not add it to any new or restarted instances. OpsWorks Stacks does not remove security groups from online instances.

> **Note**
>
> If your stack is running in a VPC, you can add or remove a security group for an online instance by using the Amazon EC2 console, API, or CLI. However, this security group will not be visible in the OpsWorks Stacks console. If you want to remove the security group, you must also use Amazon EC2. For more information, see Security Groups.

Note the following:

- You cannot restrict a built-in security group's port access settings by adding a more restrictive security group. When there are multiple security groups, Amazon EC2 uses the most permissive settings.
- You should not modify a built-in security group's configuration. When you create a stack, OpsWorks Stacks overwrites the built-in security groups' configurations, so any changes that you make will be lost the next time you create a stack.

If you discover that you need more restrictive security group settings for one or more layers take these steps:

1. Create custom security groups with appropriate settings and add them to the appropriate layers.

   Every layer in your stack must have at least one security group in addition to the built-in group, even if only one layer requires custom settings.
2. Edit the stack configuration and switch the **Use OpsWorks security groups** setting to **No**.

   OpsWorks Stacks automatically removes the built-in security group from every layer.

For more information on security groups, see Amazon EC2 Security Groups.

EC2 Instance Profile

You can change the EC2 profile for the layer's instances. For more information, see Specifying Permissions for Apps Running on EC2 instances.

CloudWatch Logs

The **CloudWatch Logs** tab lets you enable or disable Amazon CloudWatch Logs. CloudWatch Logs integration works with Chef 11.10 and Chef 12 Linux-based stacks. For more information about enabling CloudWatch Logs integration and specifying the logs that you want to manage in the CloudWatch Logs console, see Using Amazon CloudWatch Logs with OpsWorks Stacks.

Tags

The **Tags** tab lets you apply cost allocation tags to your layer. After you add tags, you can activate them in the AWS Billing and Cost Management console. When you create a tag, you are applying the tag to every resource within the tagged structure. For example, if you apply a tag to a layer,

you are applying the tag to every instance, Amazon EBS volume, or Elastic Load Balancing load balancer in the layer. For more information about how to activate your tags and use them to track and manage the costs of your OpsWorks Stacks resources, see Using Cost Allocation Tags and Activating User-Defined Cost Allocation Tags in the *Billing and Cost Management User Guide*. For more information about tagging in OpsWorks Stacks, see Tags.

Using Auto Healing to Replace Failed Instances

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Every instance has an OpsWorks Stacks agent that communicates regularly with the service. OpsWorks Stacks uses that communication to monitor instance health. If an agent does not communicate with the service for more than approximately five minutes, OpsWorks Stacks considers the instance to have failed.

Auto healing is set at the layer level; you can change the auto healing setting by editing layer settings, as shown in the following screenshot.



> **Note**
>
> An instance can be a member of multiple layers. If any of those layers has auto healing disabled, OpsWorks Stacks does not heal the instance if it fails.

If a layer has auto healing enabled—the default setting—OpsWorks Stacks automatically replaces the layer's failed instances as follows:

Instance store-backed instance

1. Stops the Amazon EC2 instance, and verifies that it has shut down.
2. Deletes the data on the root volume.
3. Creates a new Amazon EC2 instance with the same host name, configuration, and layer membership.
4. Reattaches any Amazon EBS volumes, including volumes that were attached after the old instance was originally started.
5. Assigns a new public and private IP Address.
6. If the old instance was associated with an Elastic IP address, associates the new instance with the same IP address.

Amazon EBS-backed instance

1. Stops the Amazon EC2 instance, and verifies that it has stopped.
2. Starts the EC2 instance.

After the auto-healed instance is back online, OpsWorks Stacks triggers a Configure lifecycle event on all of the stack's instances. The associated stack configuration and deployment attributes include the instance's public and private IP addresses. Custom Configure recipes can obtain the new IP addresses from the node object.

If you specify an Amazon EBS volume for a layer's instances, OpsWorks Stacks creates a new volume and attaches it to each instance when the instance is started. If you later want to detach the volume from an instance, use the Resources page.

When OpsWorks Stacks auto heals one of a layer's instances, it handles volumes in the following way:

- If the volume was attached to the instance when the instance failed, the volume and its data are saved, and OpsWorks Stacks attaches it to the new instance.
- If the volume was not attached to the instance when the instance failed, OpsWorks Stacks creates a new, empty volume with the configuration specified by the layer, and attaches that volume to the new instance.

Auto healing is enabled by default for all layers, but you can edit the layer's General Settings to disable it.

Important

If you have auto healing enabled, be sure to do the following:

- Use only the OpsWorks Stacks console, CLI, or API to stop instances.

  If you stop an instance in any other way, such as using the Amazon EC2 console, OpsWorks Stacks treats the instance as failed, and auto heals it.
- Use Amazon EBS volumes to store any data that you don't want to lose if the instance is auto healed.

  Auto healing stops the old Amazon EC2 instance, which destroys any data that is not stored on an Amazon EBS volume. Amazon EBS volumes are reattached to the new instance, which preserves any stored data.

Deleting an OpsWorks Layer

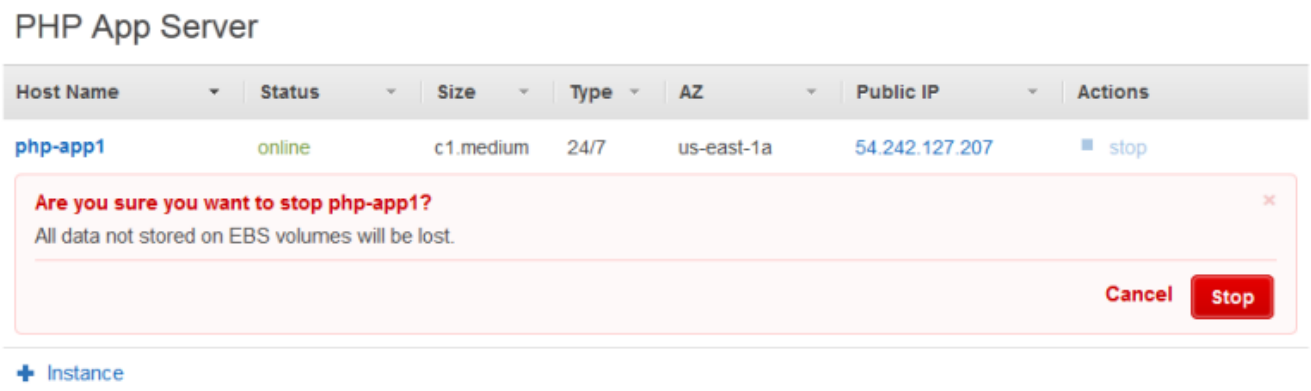
Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

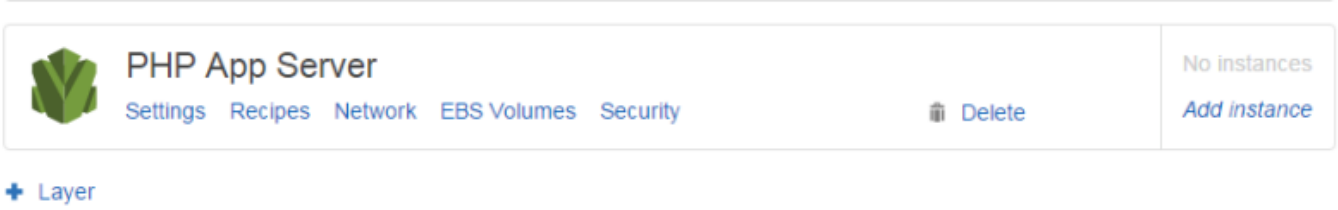
If you no longer need an OpsWorks Stacks layer, you can delete it from your stack.

To delete an OpsWorks layer

1. In the navigation pane, click **Instances**.
2. On the **Instances** page, under the name of the layer you want to delete, click **stop** in the **Actions** column for each instance.



3. After each instance has stopped, click **delete** to remove it from the layer.
4. In the navigation pane, click **Layers**.
5. On the **Layers** page, choose **Delete**.



Elastic Load Balancing Layer

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Elastic Load Balancing works somewhat differently than an OpsWorks Stacks layer. Instead of creating a layer and adding instances to it, you use the Elastic Load Balancing console or API to

create a load balancer and then attach it to an existing layer. In addition to distributing traffic to the layer's instances, Elastic Load Balancing does the following:

- Detects unhealthy Amazon EC2 instances and reroutes traffic to the remaining healthy instances until the unhealthy instances have been restored.
- Automatically scales request handling capacity in response to incoming traffic.
- If you enable connection draining, the load balancer stops sending new requests to instances that are unhealthy or about to be deregistered but keeps the connection alive, up to a specified timeout value, to allow the instance to complete any in-flight requests.

After you attach a load balancer to a layer, OpsWorks Stacks does the following:

- Deregisters any currently registered instances.
- Automatically registers the layer's instances when they come online and deregisters instances when they go offline, including load-based and time-based instances.
- Automatically starts routing requests to registered instances in their Availability Zones.

If you have enabled the load balancer's connection draining feature, you can specify whether OpsWorks Stacks supports it. If you enable connection draining support (the default setting), after an instance is shut down, OpsWorks Stacks does the following:

- Deregisters the instance from the load balancer.

  The load balancer stops sending new requests and starts connection draining.
- Delays triggering a Shutdown lifecycle event until the load balancer has completed connection draining.

If you don't enable connection draining support, OpsWorks Stacks triggers the Shutdown event as soon as the instance is shut down, even if the instance is still connected to the load balancer.

To use Elastic Load Balancing with a stack, you must first create one or more load balancers in the same region by using the Elastic Load Balancing console, CLI, or API. You should be aware of the following:

- You can attach only one load balancer to a layer.
- Each load balancer can handle only one layer.

- OpsWorks Stacks does not support Application Load Balancer. You can only use Classic Load Balancer with OpsWorks Stacks.

This means that you must create a separate Elastic Load Balancing load balancer for each layer in each stack that you want to balance and use it only for that purpose. A recommended practice is to assign a distinctive name to each Elastic Load Balancing load balancer that you plan to use with OpsWorks Stacks, such as MyStack1-RailsLayer-ELB, to avoid using a load balancer for more than one purpose.

> **Important**
>
> We recommend creating new Elastic Load Balancing load balancers for your OpsWorks Stacks layers. If you choose to use an existing Elastic Load Balancing load balancer, you should first confirm that it is not being used for other purposes and has no attached instances. After the load balancer is attached to the layer, OpsWorks removes any existing instances and configures the load balancer to handle only the layer's instances. Although it is technically possible to use the Elastic Load Balancing console or API to modify a load balancer's configuration after attaching it to a layer, you should not do so; the changes will not be permanent.

To attach an Elastic Load Balancing load balancer to a layer

1. If you have not yet done so, use the Elastic Load Balancing console, API, or CLI to create a load balancer in the stack's region. When you create the load balancer, do the following:

   - Be sure to specify a health check ping path that is appropriate for your application.

     The default ping path is `/index.html`, so if your application root does not include `index.html`, you must specify an appropriate ping path or the health check will fail.
   - If you want to use connection draining, ensure that the feature is enabled and has an appropriate timeout value.

   For more information, see Elastic Load Balancing.
2. Create the layer that you want to have balanced or edit an existing layer's Network settings.

> **Note**
>
> You cannot attach a load balancer when you create a custom layer. You must edit the layer's settings.

3. Under **Elastic Load Balancing**, select the load balancer that you want to attach to the layer and specify whether you want OpsWorks Stacks to support connection draining.

After you attach a load balancer to a layer, OpsWorks Stacks triggers a Configure lifecycle event on the stack's instances to notify them of the change. OpsWorks Stacks also triggers a Configure event when you detach a load balancer.

> **Note**
>
> After an instance has booted, OpsWorks Stacks runs the Setup and Deploy recipes, which install packages and deploy applications. After those recipes have finished, the instance is in the online state and OpsWorks Stacks registers the instance with Elastic Load Balancing. OpsWorks Stacks also triggers a Configure event after the instance comes online. This means that Elastic Load Balancing registration and the Configure recipes could run concurrently, and the instance might be registered before the Configure recipes have finished. To ensure that a recipe finishes before an instance is registered with Elastic Load Balancing, you should add the recipe to the layer's Setup or Deploy lifecycle events. For more information, see Executing Recipes.

It is sometimes useful to remove an instance from a load balancer. For example, when you update an app, we recommend that you deploy the app to a single instance and verify that the app is working properly before deploying it to every instance. You typically remove that instance from the load balancer, so it does not receive user requests until you have verified the update.

You must use the Elastic Load Balancing console or API to temporarily remove an online instance from a load balancer. The following describes how to use the console.

To temporarily remove an instance from a load balancer

1. Open the Amazon EC2 console and choose **Load Balancers**.
2. Choose the appropriate load balancer and open the **Instances** tab.

3. Choose **Remove from Load Balancer** in the instance's **Actions** column.
4. When you have finished, choose **Edit Instances**, and return the instance to the load balancer.

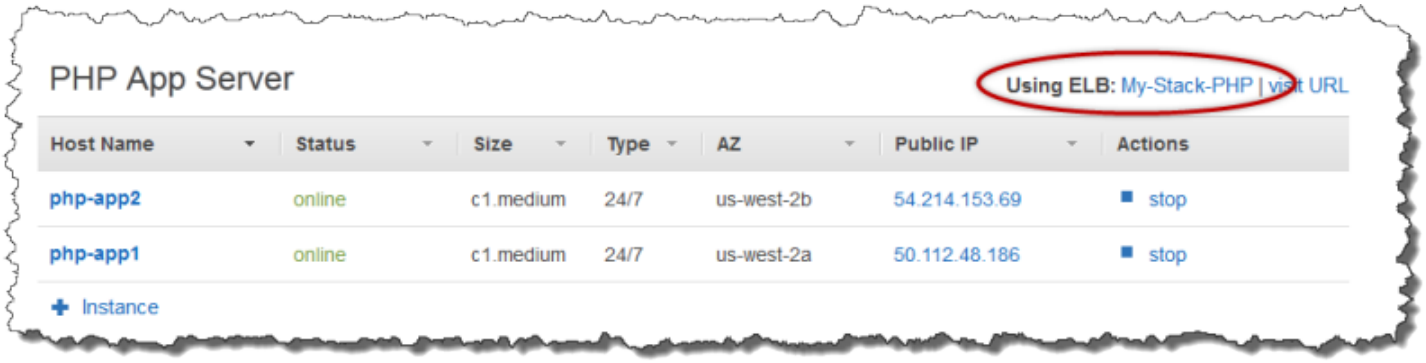
> **Important**
>
> If you use the Elastic Load Balancing console or API to remove an instance from a load balancer, you must also use Elastic Load Balancing to put it back. OpsWorks Stacks is not aware of operations that you perform with other service consoles or APIs, and it will not return the instance to the load balancer for you. Sometimes, OpsWorks Stacks can add the instance back to the ELB, but this is not guaranteed behavior and does not occur in all cases.

You can attach multiple load balancers to a particular set of instances as follows:

To attach multiple load balancers

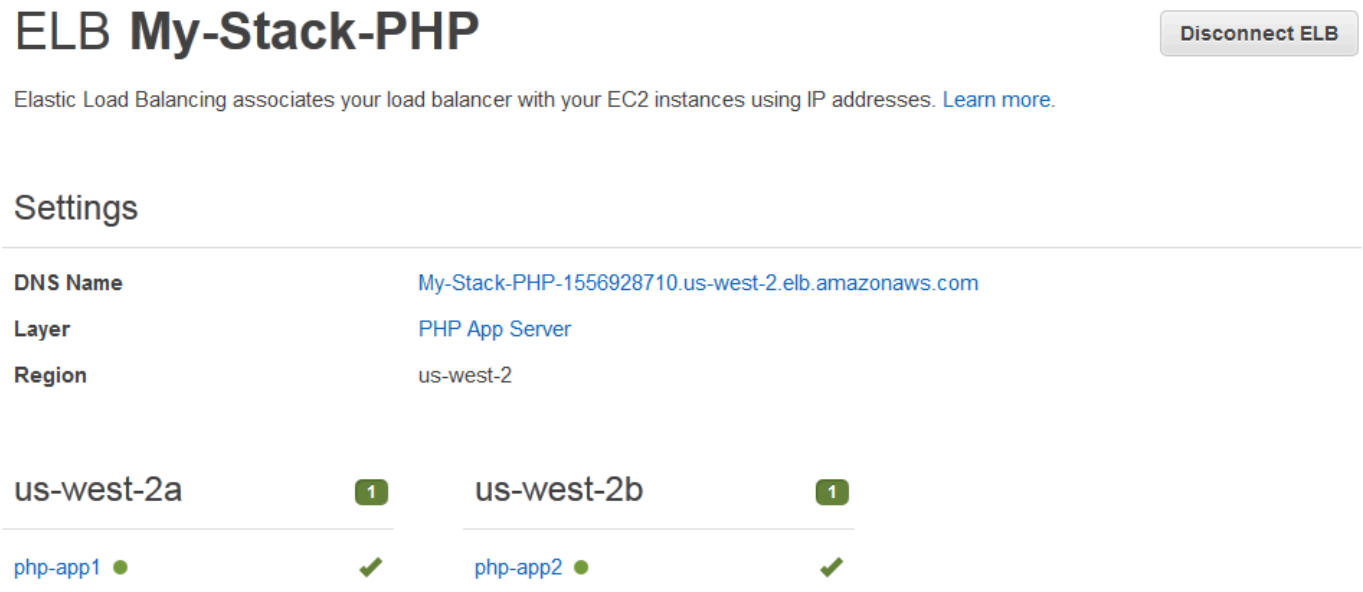
1. Use the Elastic Load Balancing console, API, or CLI to create a set of load balancers.
2. Create a custom layer for each load balancer and attach one of the load balancers to it. You don't need to implement any custom recipes for these layers; a default custom layer is sufficient.
3. Add the set of instances to each custom layer.

You can examine a load balancer's properties by going to the Instances page and clicking the appropriate load balancer name.



The **ELB** page shows the load balancer's basic properties, including its DNS name and the health status of the associated instances. If the stack is running in a VPC, the page shows subnets rather

than Availibility Zones. A green check indicates a healthy instance. You can click on the name to connect to a server, through the load balancer.



Amazon RDS Service Layer

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

An Amazon RDS service layer represents an Amazon RDS instance. The layer can represent only existing Amazon RDS instances, which you must create separately by using the Amazon RDS console or API.

The basic procedure for incorporating an Amazon RDS service layer into your stack is as follows:

1. Use the Amazon RDS console, API, or CLI to create an instance.

   Be sure to record the instance's ID, master user name, master password, and database name.
2. To add an Amazon RDS layer to your stack, register the Amazon RDS instance with the stack.

3. Attach the layer to an app, which adds the Amazon RDS instance's connection information to the app's `deploy` attributes.
4. Use the language-specific files or the information in the `deploy` attributes to connect the application to the Amazon RDS instance.

   For more information on how to connect an application to a database server, see the section called "Connecting to a Database"

> **Warning**
>
> Be sure that the characters in the instance's master password and user name are compatible with your application server. For example, with the Java App Server layer, including & in either string causes an XML parsing error that prevents the Tomcat server from starting up.

Topics

- Specifying Security Groups
- Registering an Amazon RDS Instance with a Stack
- Associating Amazon RDS Service Layers with Apps
- Removing an Amazon RDS Service Layer from a Stack

Specifying Security Groups

To use an Amazon RDS instance with OpsWorks Stacks, the database or VPC security groups must allow access from the appropriate IP addresses. For production use, a security group usually limits access to only those IP addresses that need to access the database. It typically includes the addresses of the systems that you use to manage the database and the OpsWorks Stacks instances that need to access the database. OpsWorks Stacks automatically creates an Amazon EC2 security group for each type of layer when you create your first stack in a region. A simple way to provide access for OpsWorks Stacks instances is to assign the appropriate OpsWorks Stacks security groups to the Amazon RDS instance or VPC.

To specify security groups for an existing Amazon RDS instance

1. Open the Amazon RDS console at https://console.aws.amazon.com/rds/.

2. Click **Instances** in the navigation pane and select the appropriate Amazon RDS instance. Click **Instance Actions**, **Modify**.
3. Select the following security groups from the **Security Group** list and then click **Continue** and **Modify DB Instance** to update the instance.
   - The **AWS-OpsWorks-DB-Master-Server (*security_group_id*)** security group.
   - The security group for the app server layer whose instances will be connecting to the database. The group name includes the layer name. For example, to provide database access to PHP App Server instances, specify the **AWS-OpsWorks-PHP-App-Server** group.

If you are creating a new Amazon RDS instance, you can specify the appropriate OpsWorks Stacks security groups on the Launch DB Instance wizard's **Configure Advanced Settings** page. For a description of how to use this wizard, see Creating a MySQL DB Instance and Connecting to a Database on a MySQL DB Instance.

For information on how to specify VPC security groups, see Security Groups for Your VPC.

Registering an Amazon RDS Instance with a Stack

To add an Amazon RDS service layer in a stack, you must register an instance with the stack.

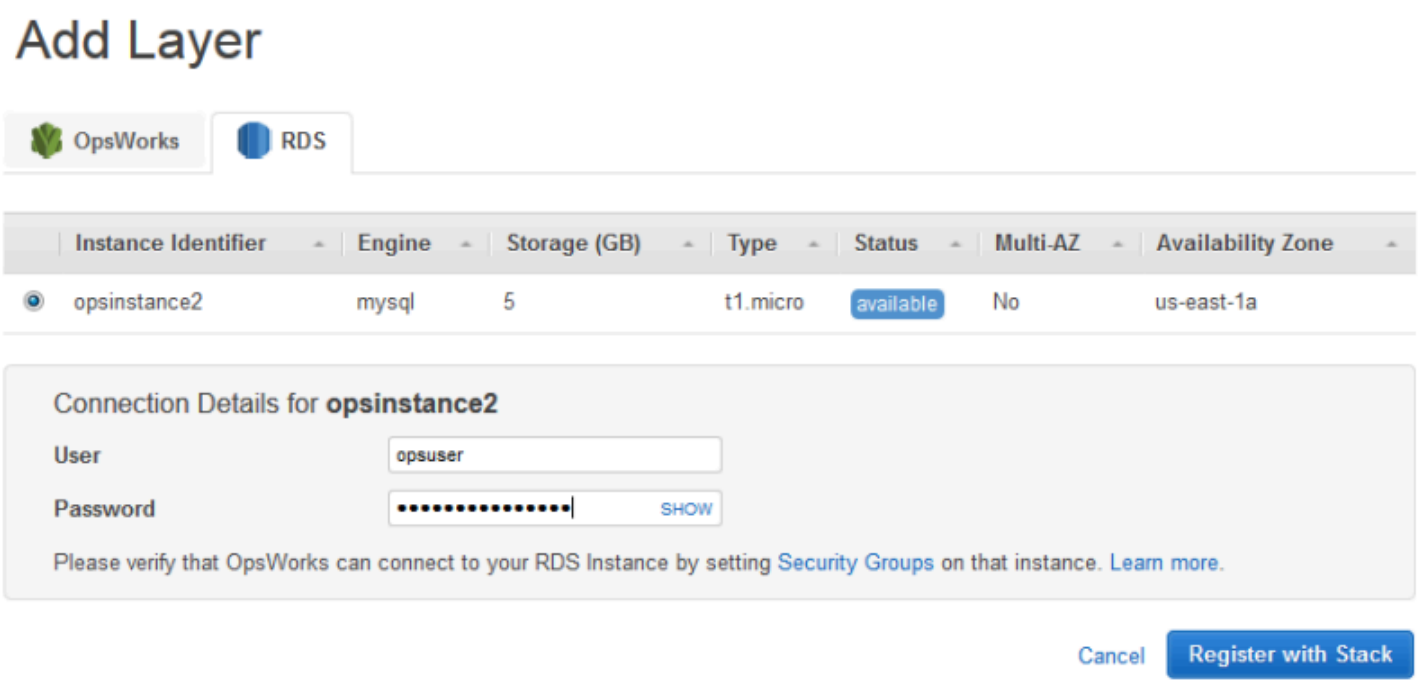
To register an Amazon RDS instance with a stack

1. In the OpsWorks Stacks console, click **Layer** in the navigation pane, click **+ Layer** or **Add a layer** to open the **Add Layer** page, and then click the **RDS** tab.
2. If necessary, update the stack's service role, as described in Updating the Stack's Service Role.
3. Click the RDS tab to list the available Amazon RDS instances.

> **Note**
>
> If your account does not have any Amazon RDS instances, you can create one by clicking **Add an RDS instance** on the RDS tab, which takes you to the Amazon RDS console and starts the **Launch a DB Instance** wizard. You can also go directly to the Amazon RDS console and click **Launch a DB Instance**, or use the Amazon RDS API or CLI. For more information on how to create an Amazon RDS instance, see Getting Started with Amazon RDS.

4. Select the appropriate instance, set **User** and **Password** to the appropriate user and password values and click **Register to Stack**.

> **Important**
>
> You must ensure that the user and password that you use to register the Amazon RDS instance correspond to a valid user and password. If they do not, your applications will not be able connect to the instance. However, you can edit the layer to provide valid user and password values and then redeploy the app.



When you add an Amazon RDS service layer to a stack, OpsWorks Stacks assigns it an ID and adds the associated Amazon RDS configuration to the stack configuration and deployment attribute's `[:opsworks][:stack]` attribute.

> **Note**
>
> If you change a registered Amazon RDS instance's password, you must manually update the password in OpsWorks Stacks and then redeploy your apps to update the stack configuration and deployment attributes on the stack's instances.

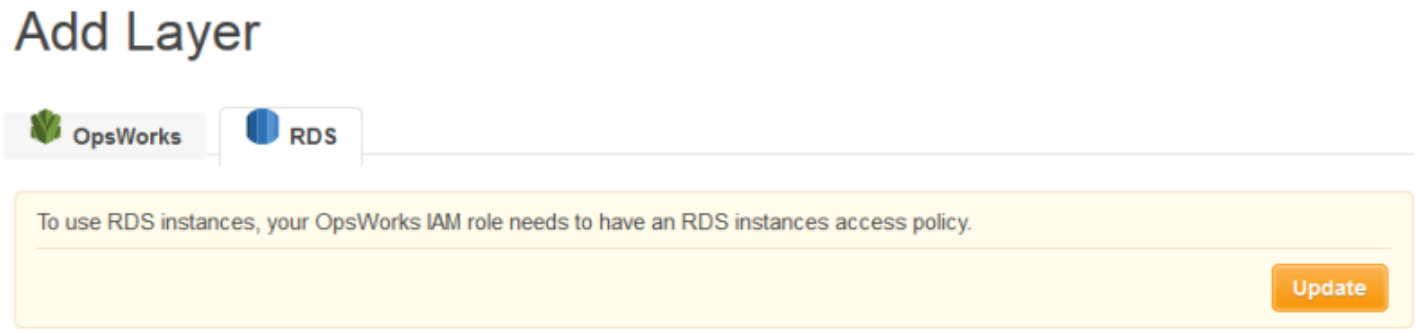
Topics

- Updating the Stack's Service Role

Updating the Stack's Service Role

Every stack has an IAM service role that specifies what actions OpsWorks Stacks can perform on your behalf with other AWS services. To register an Amazon RDS instance with a stack, its service role must grant OpsWorks Stacks permissions to access Amazon RDS.

The first time you add an Amazon RDS service layer to one of your stacks, the service role might lack the required permissions. If so, when you click the RDS tab on the **Add Layer** page, you will see the following.



Click **Update** to have OpsWorks Stacks update the service role's policy to the following.

```
{"Statement": [{"Action": ["ec2:*", "iam:PassRole",
                       "cloudwatch:GetMetricStatistics",
                       "elasticloadbalancing:*",
                       "rds:*"],
            "Effect": "Allow",
            "Resource": ["*"] }]
}
```

> **Note**
>
> You need to perform the update only once. The updated role is then automatically used by all of your stacks.

Associating Amazon RDS Service Layers with Apps

After you add an Amazon RDS service layer, you can associate it with an app.

- You can associate an Amazon RDS layer to an app when you create the app, or later by editing the app's configuration.
- To disassociate an Amazon RDS layer from an app, edit the app's configuration to specify a different database server, or no server.

  The Amazon RDS layer remains part of the stack, and can be associated with a different app.

After you associate an Amazon RDS instance with an app, OpsWorks Stacks puts the database connection information on the app's servers. The application on each server instance can then use this information to connect to the database. For more information on how to connect to an Amazon RDS instance, see the section called “Connecting to a Database”.

Removing an Amazon RDS Service Layer from a Stack

To remove an Amazon RDS service layer from a stack, you deregister it.

To deregister an Amazon RDS service layer

1. Click **Layers** in the navigation pane and click the Amazon RDS service layer's name.
2. Click **Deregister** and confirm that you want to deregister the layer.

This procedure removes the layer from the stack, but it does not delete the underlying Amazon RDS instance. The instance and any databases remain in your account and can be registered with other stacks. You must use the Amazon RDS console, API, or CLI to delete the instance. For more information, see Deleting a DB Instance.

ECS Cluster Layers

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The Amazon Elastic Container Service service (Amazon ECS) manages Docker containers on a cluster of Amazon Elastic Compute Cloud (Amazon EC2) instances, known as container instances. An ECS Cluster layer represents an Amazon ECS cluster, and simplifies cluster management by providing features that include:

- Streamlined container instance provisioning and management
- Container instance operating system and package updates
- User permissions management
- Container instance performance monitoring
- Amazon Elastic Block Store (Amazon EBS) volume management
- Public and Elastic IP address management
- Security group management

The ECS Cluster layer has the following restrictions and requirements:

- The layer is available only for Chef 11.10 or Chef 12 Linux stacks running in a VPC, including a default VPC.
- The layer's instances must be running one of the following operating systems.
  - Amazon Linux 2
  - Amazon Linux 2018.03
  - Amazon Linux 2017.09
  - Amazon Linux 2017.03
  - Amazon Linux 2016.09
  - Amazon Linux 2016.03
  - Amazon Linux 2015.09
  - Amazon Linux 2015.03
  - Ubuntu 18.04 LTS
  - Ubuntu 16.04 LTS
  - Ubuntu 14.04 LTS

- Custom
- The OpsWorks Stacks agent version on the layer's instances must be `3425-20150727112318` or later.

Topics

- Adding an ECS Cluster Layer to a Stack
- Managing the ECS Cluster
- Deleting an ECS Cluster Layer from a Stack

Adding an ECS Cluster Layer to a Stack

OpsWorks Stacks simplifies the process of launching and maintaining container instances for existing Amazon ECS clusters. To create or launch other Amazon ECS entities, such as clusters and tasks, use the Amazon ECS console, command line interface (CLI), or API. (For more information, see the Amazon Elastic Container Service Developer Guide.) You can then associate a cluster with a stack by creating an ECS Cluster layer, which you can use to manage the cluster in OpsWorks Stacks.

You can associate clusters with stacks as follows:

- Each stack can have one ECS Cluster layer, which represents a single cluster.
- A cluster can be associated with only one stack.

Before you can add ECS Cluster layers to your stacks, you must update the OpsWorks Stacks AWS Identity and Access Management (IAM) service role, which is usually named `aws-opsworks-service-role`, to allow OpsWorks Stacks to interact with Amazon ECS on your behalf. For more information on the service role, see Allowing OpsWorks Stacks to Act on Your Behalf.

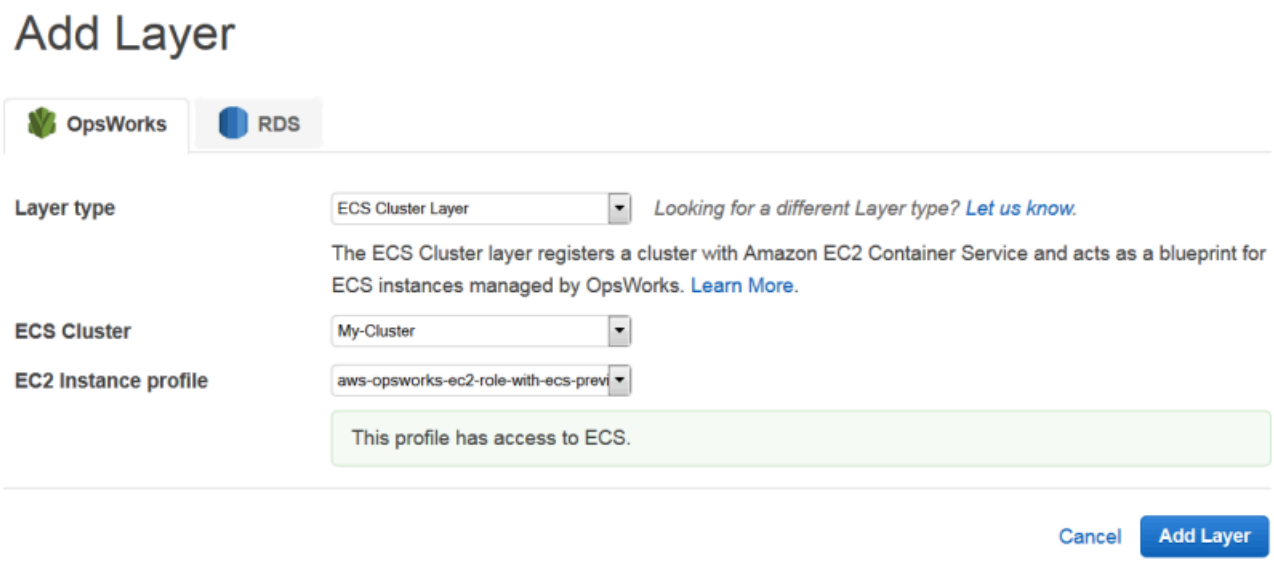
The first time you create an ECS Cluster layer, the console provides an **Update** button that you can choose to direct OpsWorks Stacks to update the role for you. OpsWorks Stacks then displays the **Add Layer** page so you can add the layer to the stack. You need to update the service role only once. You can then use the updated role to add an ECS Cluster layer to any stack.

Note

If you prefer, you can manually update the service role's policy by adding `ecs:*` permission to the existing policy, as follows:

```
{
  "Statement": [
    {
      "Action": [
        "ec2:*",
        "iam:PassRole",
        "cloudwatch:GetMetricStatistics",
        "elasticloadbalancing:*",
        "rds:*",
        "ecs:*"
      ],
      "Effect": "Allow",
      "Resource": ["*"]
    }
  ]
}
```

Associating a cluster with a stack requires two operations: registering the cluster with the stack and then creating the associated layer. The OpsWorks Stacks console combines these steps; layer creation automatically registers the specified cluster. If you use the OpsWorks Stacks API, CLI, or SDK, you must use separate operations to register the cluster and create the associated layer. To use the console to add an ECS Cluster layer to your stack, choose **Layers**, choose **+Layer** or **Add a Layer**, and then chose the ECS Cluster layer type.



The **Add Layer** page includes the following configuration options:

ECS Cluster

The Amazon ECS cluster that you want to register with the stack.

EC2 Instance profile

The cluster's Amazon Elastic Compute Cloud(Amazon EC2) instance profile. This profile grants permission for applications running on the cluster's container instances to access other AWS services, including Amazon ECS. When you create your first ECS Cluster layer, choose **New profile with ECS access** to direct OpsWorks Stacks to create the required profile, which is named `aws-opsworks-ec2-role-with-ecs`. You can then use that profile for all subsequent ECS Cluster layers. For more information on the instance profile, see Specifying Permissions for Apps Running on EC2 instances.

You can specify other settings by editing the layer's configuration, including:

- Attaching an Elastic Load Balancing load balancer to the layer.

  This approach might be suitable for some use cases, but Amazon ECS provides more sophisticated options. For more information, see Service Load Balancing.
- Specifying whether to automatically assign public IP addresses or Elastic IP addresses to the container instances.

If you disable automatic assignment for both address types, the instance will not come online unless the subnet has a properly configured NAT. For more information, see Running a Stack in a VPC.

Managing the ECS Cluster

After you create an ECS Cluster layer, you can use OpsWorks Stacks to manage the cluster as follows:

Provision and manage container instances

Initially, an ECS Cluster layer does not include any container instances, even if the original cluster did. One option is to manage the layer's instances by using an appropriate combination of the following:

- Manually add 24/7 instances to the layer and delete them when they are no longer needed.
- Add or delete instances on a schedule by adding time-based instances to the layer.
- Add or delete instances based on OpsWorks Stacks host metrics or CloudWatch alarms by adding load-based instances to the layer.

> **Note**
>
> If Amazon ECS is not supported for the stack's default operating system, you must explicitly specify a supported operating system—Amazon Linux 2, Amazon Linux 2018.03, Amazon Linux 2017.09, Amazon Linux 2017.03, Amazon Linux 2016.09, Amazon Linux 2016.03, Amazon Linux 2015.09, Amazon Linux 2015.03, Ubuntu 18.04 LTS, Ubuntu 16.04 LTS, Ubuntu 14.04 LTS, or Custom—when you create the container instances. Do not use the ECS Optimized AMI to create instances in an ECS layer, because this AMI already includes the ECS agent. OpsWorks Stacks also attempts to install the ECS agent during the instance setup process, and the conflict can cause setup to fail.

For more information, see Optimizing the Number of Servers. OpsWorks Stacks assigns the **AWS-OpsWorks-ECS-Cluster** security group to each instance. After each new instance finishes booting, OpsWorks Stacks converts it into a container instance by installing Docker and the Amazon ECS agent, and then registering the instance with the cluster.

If you prefer to use existing container instances, you can register them with the stack and assign them to the ECS Cluster layer. Note that the instances must be running a supported operating system, Amazon Linux 2015.03 or later, or Ubuntu 14.04 LTS or later.

> **Note**
>
> A container instance cannot belong to both an ECS Cluster layer and another built-in layer. However, a container instance *can* belong to an ECS Cluster layer and one or more custom layers.

Run operating system and package updates

After a new instance finishes booting, OpsWorks Stacks installs the latest updates. You can then use OpsWorks Stacks to keep the container instances up to date. For more information, see Managing Security Updates.

Manage user permissions

OpsWorks Stacks provides a simple way to manage permissions on the container instances, including managing users' SSH keys. For more information, see Managing User Permissions and Managing SSH Access.

Monitor performance metrics

OpsWorks Stacks provides a variety of ways to monitor performance metrics for the stack, layer, or individual instances. For more information, see Monitoring.

You handle other management tasks, such as creating tasks or services, through Amazon ECS. For more information, see the Amazon Elastic Container Service Developer Guide.

> **Note**
>
> To go directly to the cluster's page on the Amazon ECS console, choose **Instances**, and then choose **ECS Cluster**, which is near the upper right corner of the ECS Cluster layer's section.

Deleting an ECS Cluster Layer from a Stack

When you no longer need the cluster, delete the ECS Cluster layer and deregister the associated cluster. Removing a cluster from a stack requires two operations: deregistering the cluster and then

deleting the associated layer. The OpsWorks Stacks console combines these steps; layer deletion automatically deregisters the specified cluster. If you use the OpsWorks Stacks API, CLI, or SDK, you must use separate operations to deregister the cluster and delete the associated layer.

To use the console to delete an ECS Cluster layer

1. If you want to control how tasks are shut down, use the Amazon ECS console, API, or CLI to scale down and delete the cluster's services. For more information, see Cleaning Up Your Amazon ECS Resources.
2. Stop the layer's instances, and then delete them. When you stop a container instance, OpsWorks Stacks automatically stops any running tasks, deregisters the instance from the cluster, and terminates the instance.

   > **Note**
   >
   > If you have registered existing container instances with the stack, you can unassign the instances from the layer and then deregister them, which returns the instances to ECS control.

3. Delete the layer. OpsWorks Stacks deregisters the associated cluster, but does not delete it. The cluster remains in Amazon ECS.

Custom OpsWorks Stacks Layers

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

A custom layer has only a minimal set of recipes. You then add appropriate functionality to the layer by implementing custom recipes and assigning them to the layer's lifecycle events.

The custom layer has the following configuration settings.

> **Note**
>
> OpsWorks Stacks automatically installs Ruby on the layer's instances. If you want to run Ruby code on the instance but don't want to use the default Ruby version, you can use custom JSON or a custom attributes file to specify your preferred version. For more information, see Ruby Versions.

The basic procedure for creating a custom layer has the following steps:

1. Implement a cookbook that contains the recipes and associated files required to install and configure packages, handle configuration changes, deploy apps, and so on.

   Depending on your requirements, you might also need recipes to handle undeployment and shutdown tasks. For more information, see Cookbooks and Recipes.
2. Create a custom layer.
3. Assign your recipes to the appropriate lifecycle events.

You then add instances to the layer, start them, and deploy apps to those instances.

> **Important**
>
> To deploy apps to a custom layer's instances, you must implement recipes to handle the deploy operation and assign them to the layer's Deploy event.

Per-layer Operating System Package Installations

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Starting with Chef 12, you must use custom recipes to install packages on layers that are running different operating systems. This approach provides you with maximum flexibility and control over package installations.

For example, suppose that you want to install Apache on layers that are running RedHat, Ubuntu, and Amazon versions of the Linux operating system. The Apache package for RedHat and Amazon Linux is called `httpd`, but on Ubuntu, it is called `apache2`.

To address the difference in package naming, you can use syntax similar to that in the following example recipe. The recipe installs the Apache package appropriate for each operating system. This example is based on the Chef documentation.

```
package "Install Apache" do
   case node[:platform]
      when "redhat", "amazon"
         package_name "httpd"
      when "ubuntu"
         package_name "apache2"
   end
end
```

For detailed information on how to use the `package` resource to manage packages, go to the package page in the Chef documentation.

Alternatively, you can use the `value_for_platform` helper method from the Chef Recipe DSL (domain-specific language), which accomplishes the same thing more succinctly:

```
package "Install Apache" do
   package_name value_for_platform(
      ["redhat", "amazon"] => { "default" => "httpd" },
      ["ubuntu"] => { "default" => "apache2" }
   )
end
```

For information on using the `value_for_platform` helper method, go to About the Recipe DSL.

Instances

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

An instance represents a computing resource, such as an Amazon EC2 instance, which handles the work of serving applications, balancing traffic, and so on. An instance's operating system can have any of several Linux distributions, or Windows Server 2012 R2.

You can add instances to a stack in either of the following ways:

- Use OpsWorks Stacks to add instances to a stack. The instances that you add represent Amazon EC2 instances.
- For Linux-based stacks, you can register instances that were created elsewhere—including instances that you created with Amazon EC2 and *on-premises* instances that are running on your own hardware.

  You can then use OpsWorks Stacks to manage these instances in much the same way as instances created with OpsWorks Stacks

This section describes how to use OpsWorks Stacks to create and manage instances.

Topics

- Using OpsWorks Stacks Instances
- Using Computing Resources Created Outside of OpsWorks Stacks
- Editing the Instance Configuration
- Deleting OpsWorks Stacks Instances
- Using SSH to Log In to a Linux Instance
- Using RDP to Log In to a Windows Instance

Using OpsWorks Stacks Instances

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can use OpsWorks Stacks to create instances and add them to the stack.

Topics

- OpsWorks Stacks operating systems
- Adding an Instance to a Layer
- Using Custom AMIs
- Manually Starting, Stopping, and Rebooting 24/7 Instances
- Managing load with time-based and load-based instances

OpsWorks Stacks operating systems

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks supports the 64-bit versions of several built-in operating systems, including Amazon and Ubuntu Linux distributions, and Microsoft Windows Server. Some general notes:

- A stack's instances can run either Linux or Windows.

A stack can have different Linux versions or distributions on different instances, but you cannot mix Linux and Windows instances.

- You can use custom AMIs (Amazon Machine Images), but they must be based on one of the OpsWorks Stacks-supported AMIs that are described in topics in this section. While it might be possible to create or register instances with other operating systems (such as CentOS 6.*x*) that have been created from custom or community-generated AMIs, these are not officially supported.
  - Linux operating systems
  - Microsoft Windows Server
- You can start and stop instances manually or have OpsWorks Stacks automatically scale the number of instances.

  You can use time-based automatic scaling with any stack; Linux stacks also can use load-based scaling.
- In addition to using OpsWorks Stacks to create Amazon EC2 instances, you can also register instances with a Linux stack that were created outside of OpsWorks Stacks.

  This includes Amazon EC2 instances and instances running on your own hardware. However, they must be running one of the supported Linux distributions. You cannot register Amazon EC2 or on-premises Windows instances.

You can run the OpsWorks Stacks `DescribeOperatingSystems` API to return a list of supported operating systems and their supported versions of Chef. The following is an example command, using the AWS CLI.

```
aws opsworks describe-operating-systems
```

The following is an example response.

```
{
    "OperatingSystems": [
        {
            "Name": "Amazon Linux",
            "Id": "Amazon Linux",
            "Type": "Linux",
            "ConfigurationManagers": [
                {
```

```
                    "Name": "Chef",
                    "Version": "11.10"
                },
                {
                    "Name": "Chef",
                    "Version": "11.4"
                },
                {
                    "Name": "Chef",
                    "Version": "0.9"
                }
            ],
            "ReportedName": "amazon",
            "ReportedVersion": "2014.03",
            "Supported": false
        },
        {
            "Name": "Amazon Linux 2",
            "Id": "Amazon Linux 2",
            "Type": "Linux",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12"
                }
            ],
            "ReportedName": "amazon",
            "ReportedVersion": "2"
        },
        {
            "Name": "Amazon Linux 2014.09",
            "Id": "Amazon Linux 2014.09",
            "Type": "Linux",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "11.10"
                },
                {
                    "Name": "Chef",
                    "Version": "11.4"
                },
                {
                    "Name": "Chef",
```

```
                    "Version": "0.9"
                }
            ],
            "ReportedName": "amazon",
            "ReportedVersion": "2014.09",
            "Supported": false
        },
        {
            "Name": "Amazon Linux 2015.03",
            "Id": "Amazon Linux 2015.03",
            "Type": "Linux",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12"
                },
                {
                    "Name": "Chef",
                    "Version": "11.10"
                },
                {
                    "Name": "Chef",
                    "Version": "11.4"
                },
                {
                    "Name": "Chef",
                    "Version": "0.9"
                }
            ],
            "ReportedName": "amazon",
            "ReportedVersion": "2015.03",
            "Supported": false
        },
        {
            "Name": "Amazon Linux 2015.09",
            "Id": "Amazon Linux 2015.09",
            "Type": "Linux",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12"
                },
                {
                    "Name": "Chef",
```

```
                    "Version": "11.10"
                },
                {
                    "Name": "Chef",
                    "Version": "11.4"
                },
                {
                    "Name": "Chef",
                    "Version": "0.9"
                }
            ],
            "ReportedName": "amazon",
            "ReportedVersion": "2015.09",
            "Supported": false
        },
        {
            "Name": "Amazon Linux 2016.03",
            "Id": "Amazon Linux 2016.03",
            "Type": "Linux",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12"
                },
                {
                    "Name": "Chef",
                    "Version": "11.10"
                },
                {
                    "Name": "Chef",
                    "Version": "11.4"
                },
                {
                    "Name": "Chef",
                    "Version": "0.9"
                }
            ],
            "ReportedName": "amazon",
            "ReportedVersion": "2016.03"
        },
        {
            "Name": "Amazon Linux 2016.09",
            "Id": "Amazon Linux 2016.09",
            "Type": "Linux",
```

```
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12"
                },
                {
                    "Name": "Chef",
                    "Version": "11.10"
                },
                {
                    "Name": "Chef",
                    "Version": "11.4"
                },
                {
                    "Name": "Chef",
                    "Version": "0.9"
                }
            ],
            "ReportedName": "amazon",
            "ReportedVersion": "2016.09"
        },
        {
            "Name": "Amazon Linux 2017.03",
            "Id": "Amazon Linux 2017.03",
            "Type": "Linux",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12"
                },
                {
                    "Name": "Chef",
                    "Version": "11.10"
                },
                {
                    "Name": "Chef",
                    "Version": "11.4"
                },
                {
                    "Name": "Chef",
                    "Version": "0.9"
                }
            ],
            "ReportedName": "amazon",
```

```
            "ReportedVersion": "2017.03"
        },
        {
            "Name": "Amazon Linux 2017.09",
            "Id": "Amazon Linux 2017.09",
            "Type": "Linux",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12"
                },
                {
                    "Name": "Chef",
                    "Version": "11.10"
                },
                {
                    "Name": "Chef",
                    "Version": "11.4"
                },
                {
                    "Name": "Chef",
                    "Version": "0.9"
                }
            ],
            "ReportedName": "amazon",
            "ReportedVersion": "2017.09"
        },
        {
            "Name": "Amazon Linux 2018.03",
            "Id": "Amazon Linux 2018.03",
            "Type": "Linux",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12"
                },
                {
                    "Name": "Chef",
                    "Version": "11.10"
                }
            ],
            "ReportedName": "amazon",
            "ReportedVersion": "2018.03"
        },
```

```
        {
            "Name": "CentOS Linux 7",
            "Id": "CentOS Linux 7",
            "Type": "Linux",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12"
                }
            ],
            "ReportedName": "CentOS Linux",
            "ReportedVersion": "7"
        },
        {
            "Name": "Microsoft Windows Server 2012 R2 Base",
            "Id": "Microsoft Windows Server 2012 R2 Base",
            "Type": "Windows",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12.2"
                }
            ],
            "ReportedName": "microsoft windows server",
            "ReportedVersion": "2012 r2 standard",
            "Supported": false
        },
        {
            "Name": "Microsoft Windows Server 2012 R2 with SQL Server Express",
            "Id": "Microsoft Windows Server 2012 R2 with SQL Server Express",
            "Type": "Windows",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12.2"
                }
            ],
            "ReportedName": "microsoft windows server",
            "ReportedVersion": "2012 r2 standard",
            "Supported": false
        },
        {
            "Name": "Microsoft Windows Server 2012 R2 with SQL Server Standard",
            "Id": "Microsoft Windows Server 2012 R2 with SQL Server Standard",
```

```
            "Type": "Windows",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12.2"
                }
            ],
            "ReportedName": "microsoft windows server",
            "ReportedVersion": "2012 r2 standard",
            "Supported": false
        },
        {
            "Name": "Microsoft Windows Server 2012 R2 with SQL Server Web",
            "Id": "Microsoft Windows Server 2012 R2 with SQL Server Web",
            "Type": "Windows",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12.2"
                }
            ],
            "ReportedName": "microsoft windows server",
            "ReportedVersion": "2012 r2 standard",
            "Supported": false
        },
        {
            "Name": "Microsoft Windows Server 2019 Base",
            "Id": "Microsoft Windows Server 2019 Base",
            "Type": "Windows",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12.2"
                }
            ],
            "ReportedName": "microsoft windows server",
            "ReportedVersion": "2019 datacenter"
        },
        {
            "Name": "Microsoft Windows Server 2019 with SQL Server Express",
            "Id": "Microsoft Windows Server 2019 with SQL Server Express",
            "Type": "Windows",
            "ConfigurationManagers": [
                {
```

```
                "Name": "Chef",
                "Version": "12.2"
            }
        ],
        "ReportedName": "microsoft windows server",
        "ReportedVersion": "2019 datacenter"
    },
    {
        "Name": "Microsoft Windows Server 2019 with SQL Server Standard",
        "Id": "Microsoft Windows Server 2019 with SQL Server Standard",
        "Type": "Windows",
        "ConfigurationManagers": [
            {
                "Name": "Chef",
                "Version": "12.2"
            }
        ],
        "ReportedName": "microsoft windows server",
        "ReportedVersion": "2019 datacenter"
    },
    {
        "Name": "Microsoft Windows Server 2019 with SQL Server Web",
        "Id": "Microsoft Windows Server 2019 with SQL Server Web",
        "Type": "Windows",
        "ConfigurationManagers": [
            {
                "Name": "Chef",
                "Version": "12.2"
            }
        ],
        "ReportedName": "microsoft windows server",
        "ReportedVersion": "2019 datacenter"
    },
    {
        "Name": "Microsoft Windows Server 2022 Base",
        "Id": "Microsoft Windows Server 2022 Base",
        "Type": "Windows",
        "ConfigurationManagers": [
            {
                "Name": "Chef",
                "Version": "12.2"
            }
        ],
        "ReportedName": "microsoft windows server",
```

```
            "ReportedVersion": "2022 datacenter"
        },
        {
            "Name": "Microsoft Windows Server 2022 with SQL Server Express",
            "Id": "Microsoft Windows Server 2022 with SQL Server Express",
            "Type": "Windows",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12.2"
                }
            ],
            "ReportedName": "microsoft windows server",
            "ReportedVersion": "2022 datacenter"
        },
        {
            "Name": "Microsoft Windows Server 2022 with SQL Server Standard",
            "Id": "Microsoft Windows Server 2022 with SQL Server Standard",
            "Type": "Windows",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12.2"
                }
            ],
            "ReportedName": "microsoft windows server",
            "ReportedVersion": "2022 datacenter"
        },
        {
            "Name": "Microsoft Windows Server 2022 with SQL Server Web",
            "Id": "Microsoft Windows Server 2022 with SQL Server Web",
            "Type": "Windows",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12.2"
                }
            ],
            "ReportedName": "microsoft windows server",
            "ReportedVersion": "2022 datacenter"
        },
        {
            "Name": "Red Hat Enterprise Linux 7",
            "Id": "Red Hat Enterprise Linux 7",
```

```
            "Type": "Linux",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12"
                },
                {
                    "Name": "Chef",
                    "Version": "11.10"
                }
            ],
            "ReportedName": "Red Hat Enterprise Linux",
            "ReportedVersion": "7"
        },
        {
            "Name": "Ubuntu 12.04 LTS",
            "Id": "Ubuntu 12.04 LTS",
            "Type": "Linux",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12"
                },
                {
                    "Name": "Chef",
                    "Version": "11.10"
                },
                {
                    "Name": "Chef",
                    "Version": "11.4"
                },
                {
                    "Name": "Chef",
                    "Version": "0.9"
                }
            ],
            "ReportedName": "ubuntu",
            "ReportedVersion": "12.04",
            "Supported": false
        },
        {
            "Name": "Ubuntu 14.04 LTS",
            "Id": "Ubuntu 14.04 LTS",
            "Type": "Linux",
```

```
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12"
                },
                {
                    "Name": "Chef",
                    "Version": "11.10"
                }
            ],
            "ReportedName": "ubuntu",
            "ReportedVersion": "14.04"
        },
        {
            "Name": "Ubuntu 16.04 LTS",
            "Id": "Ubuntu 16.04 LTS",
            "Type": "Linux",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12"
                }
            ],
            "ReportedName": "ubuntu",
            "ReportedVersion": "16.04"
        },
        {
            "Name": "Ubuntu 18.04 LTS",
            "Id": "Ubuntu 18.04 LTS",
            "Type": "Linux",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12"
                }
            ],
            "ReportedName": "ubuntu",
            "ReportedVersion": "18.04"
        },
        {
            "Name": "Ubuntu 20.04 LTS",
            "Id": "Ubuntu 20.04 LTS",
            "Type": "Linux",
            "ConfigurationManagers": [
```

```
                {
                    "Name": "Chef",
                    "Version": "12"
                }
            ],
            "ReportedName": "ubuntu",
            "ReportedVersion": "20.04"
        },
        {
            "Name": "Custom",
            "Id": "Custom",
            "Type": "Linux",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12"
                },
                {
                    "Name": "Chef",
                    "Version": "11.10"
                },
                {
                    "Name": "Chef",
                    "Version": "11.4"
                },
                {
                    "Name": "Chef",
                    "Version": "0.9"
                }
            ]
        },
        {
            "Name": "CustomWindows",
            "Id": "CustomWindows",
            "Type": "Windows",
            "ConfigurationManagers": [
                {
                    "Name": "Chef",
                    "Version": "12.2"
                }
            ]
        }
    ]
```

```
}
```

Topics

- Linux operating systems
- Microsoft Windows Server

Linux operating systems

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks supports the 64-bit versions of the following Linux operating systems.

- Amazon Linux and Amazon Linux 2 (see the OpsWorks Stacks console for the currently supported versions)
- Ubuntu 20.04 LTS
- CentOS 7
- Red Hat Enterprise Linux 7

You can also use custom AMIs based on these operating systems.

Some general notes on Linux instances:

Supported package versions

The supported versions and patch levels for packages, such as Ruby, depend on the operating system and version as described in the following sections.

Updates

By default, OpsWorks Stacks ensures that Linux instances have the latest security patches by automatically calling `yum update` or `apt-get update` after an instance boots.

To disable automatic updates use the CreateInstance, UpdateInstance, CreateLayer, or UpdateLayer actions—or the equivalent AWS SDK methods or AWS CLI commands— to set the `InstallUpdatesOnBoot` parameter to `false`.

To avoid service interruptions, OpsWorks Stacks does not automatically install updates after an instance is online. You can manually update an online instance's operating system at any time by running the Upgrade Operating System stack command. For more information on how to manage security updates, see Managing Security Updates.

For more control over how OpsWorks Stacks updates your instances, create a custom AMI based on one of the supported operating systems. For example, with custom AMIs you can specify which package versions are installed on an instance. Each Linux distribution has different support timelines and package-merge policies, so you should consider which approach best suits your requirements. For more information, see Using Custom AMIs.

Hosts file

Each online instance has a `/etc/hosts` file that maps IP addresses to host names. OpsWorks Stacks includes the public and private addresses for all of the stack's online instances in each instance's `hosts` file. For example, suppose that you have a stack with two Node.js App Server instances, nodejs-app1 and nodejs-app2, and one MySQL instance, db-master1. The nodejs-app1 instance's `hosts` file will look something like the following example, and the other instances' will have similar `hosts` files.

```
...
# OpsWorks Layer State
192.0.2.0 nodejs-app1.localdomain nodejs-app1
10.145.160.232 db-master1
198.51.100.0 db-master1-ext
10.243.77.78 nodejs-app2
203.0.113.0 nodejs-app2-ext
10.84.66.6 nodejs-app1
192.0.2.0 nodejs-app1-ext
```

OpsWorks Stacks agent proxy support

The OpsWorks Stacks agent for Chef 11.10 and later stacks includes basic support for proxy servers, which are typically used with isolated VPCs. To enable proxy server support, an instance must have an `/etc/environment` file that provides the appropriate settings for HTTP and

HTTPS traffic. The file should look similar to the following, where you replace the highlighted text with your proxy server's URL and port:

```
http_proxy="http://myproxy.example.com:8080/"
https_proxy="http://myproxy.example.com:8080/"
no_proxy="169.254.169.254"
```

To enable proxy support, we recommend creating a custom AMI that includes an appropriate `/etc/environment` file and using that AMI to create your instances.

> **Note**
>
> We do not recommend using a custom recipe to create an `/etc/environment` file on your instances. OpsWorks Stacks needs the proxy server data early in the setup process, before any custom recipes have executed.

Topics

- Amazon Linux
- Ubuntu LTS
- CentOS
- Red Hat Enterprise Linux

Amazon Linux

OpsWorks Stacks supports the 64-bit versions of Amazon Linux and Amazon Linux 2. In addition to regular updates and patches, Amazon Linux releases a new version approximately every six months, which can involve significant changes. When you create a stack or a new instance, you must specify which Amazon Linux version to use. When AWS releases a new version, your instances continue to run the specified version until you explicitly change it. After a new Amazon Linux version is released, there is a four-week migration period, during which AWS continues to provide regular updates for the old version. After the migration period ends, your instances can continue to run the old version, but AWS does not provide further updates. For more information, see Amazon Linux AMI FAQs.

When a new Amazon Linux version is released, we recommend that you update to the new version within the migration period so your instances continue to receive security updates. Before updating your production stack's instances, we recommend you start a new instance and verify that your app runs correctly on the new version. You can then update the production stack instances.

> **Note**
>
> By default, custom AMIs based on Amazon Linux are automatically updated to the new version when it is released. The recommended practice is to lock your custom AMI to a specific Amazon Linux version so you can defer the update until you have tested the new version. For more information, see How do I lock my AMI to a specific version?.
> If you use an AWS CloudFormation template to create stacks with instances running Amazon Linux, the templates should explicitly specify an Amazon Linux version. In particular, if your template specifies `Amazon Linux`, the instances will continue to run version 2016.09. For more information, see AWS::OpsWorks::Stack and AWS::OpsWorks::Instance.

To update an instance's Amazon Linux version, do one of the following:

- For online instances, run the **Upgrade Operating System** stack command.

  When a new Amazon Linux version is available, the **Instances** and **Stack** pages display a notice with a link that takes you to the **Run Command** page. You can then run **Upgrade Operating System** to upgrade your instance.
- For offline Amazon Elastic Block Store-backed (EBS-backed) instances, start the instances and run **Upgrade Operating System**, as described in the preceding statement.
- For offline instance store-backed instances, including time-based and load-based instances, edit the instance's **Operating system** setting to specify the new version.

  OpsWorks Stacks automatically updates the instances to the new version when they are restarted.

Amazon Linux: Supported Node.js Versions

Amazon Linux Version	Node.js Versions
2	(Not applicable to operating systems that are available for Chef 12 and higher stacks only)
2018.03	0.12.18
2017.09	0.12.18
2017.03	0.12.18
2016.09	0.12.18 0.12.17 0.12.16 0.12.15
2016.03	0.12.18 0.12.17 0.12.16 0.12.15 0.12.14 0.12.13 0.12.12 0.12.10

Amazon Linux: Supported Chef Versions

Chef Version	Supported Amazon Linux Versions
12	Amazon Linux 2 Amazon Linux 2018.03 Amazon Linux 2017.09 Amazon Linux 2017.03 Amazon Linux 2016.09

Chef Version	Supported Amazon Linux Versions
	Amazon Linux 2016.03
11.10	Amazon Linux 2018.03 Amazon Linux 2017.09 Amazon Linux 2017.03 Amazon Linux 2016.09 Amazon Linux 2016.03
11.4 (deprecated)	Amazon Linux 2016.09 Amazon Linux 2016.03

Important

Before updating t1.micro instances, make sure they have a temporary swap file, `/var/swapfile`. The t1.micro instances on Chef 0.9 stacks do not have a swap file. For Chef 11.4 and Chef 11.10 stacks, recent versions of the instance agent automatically create a swap file for t1.micro instances. However, this change was introduced over a period of several weeks, so you should check for the existence of `/var/swapfile` on instances created before approximately Mar. 24, 2014.
For t1.micro instances that lack a swap file, you can create one as follows:

- For Chef 11.10 and later stacks, create new t1.micro instances, which automatically have a swap file.
- For Chef 0.9 stacks, run the following commands on each instance as root user.

```
dd if=/dev/zero of=/var/swapfile bs=1M count=256
 mkswap /var/swapfile
 chown root:root /var/swapfile
 chmod 0600 /var/swapfile
 swapon /var/swapfile
```

You can also use these commands on Chef 11.10 and later stacks if you don't want to create new instances.

Ubuntu LTS

Ubuntu releases a new Ubuntu LTS version approximately every two years and supports each release for approximately five years. Ubuntu provides security patches and updates for the duration of the operating system support. For more information, see LTS - Ubuntu Wiki.

- You cannot update an existing Ubuntu instance to a newer release of Ubuntu.

  You must create a new Ubuntu instance and delete the older instance.
- Ubuntu 20.04 LTS is supported only for Chef 12 and higher stacks.

CentOS

OpsWorks Stacks supports the 64-bit version of CentOS 7. The initial supported version is CentOS 7, and CentOS releases a new version approximately every two years.

When you start a new instance in a CentOS stack, OpsWorks Stacks automatically installs the most current CentOS version. Because OpsWorks Stacks does not automatically update the operating system on existing instances when a new CentOS minor version is released, a newly created instance might receive a more recent version than the stack's existing instances. To keep versions consistent across your stack, you can update your existing instances to the current CentOS version, as follows:

- For online instances, run the **Upgrade Operating System** stack command, which runs `yum update` on the specified instances to update them to the current version.

  When a new CentOS 7 minor version is available, the **Instances** and **Stack** pages display a notice with a link that takes you to the **Run Command** page. You can then run **Upgrade Operating System** to upgrade your instances.
- For offline Amazon EBS-backed instances, start the instances and run **Upgrade Operating System** as described in the preceding list item.
- For offline instance store-backed instances, OpsWorks Stacks automatically installs the new version when the instances are restarted.

CentOS: Supported Chef Versions

Chef Version	Supported CentOS Version
12	CentOS 7
11.10	(None supported)
11.4 (deprecated)	(None supported)

> **Note**
>
> OpsWorks Stacks supports Apache 2.4 for CentOS instances.

Red Hat Enterprise Linux

OpsWorks Stacks supports the 64-bit version of Red Hat Enterprise Linux 7 (RHEL 7). The initial supported version is RHEL 7.1 and Red Hat releases a new minor version approximately every 9 months. Minor versions should be compatible with RHEL 7.0. For more information, see Life Cycle and Update Policies.

When you start a new instance, OpsWorks Stacks automatically installs the current RHEL 7 version. Because OpsWorks Stacks does not automatically update the operating system on existing instances when a new RHEL 7 minor version is released, a newly created instance might receive a more recent version than the stack's existing instances. To keep versions consistent across your stack, you can update your existing instances to the current RHEL 7 version, as follows:

- For online instances, run the **Upgrade Operating System** stack command, which runs `yum update` on the specified instances to update them to the current version.

  When a new RHEL 7 version is available, the **Instances** and **Stack** pages display a notice with a link that takes you to the **Run Command** page. You can then run **Upgrade Operating System** to upgrade your instances.
- For offline Amazon EBS-backed instances, start the instances and run **Upgrade Operating System** as described in the preceding list item.

- For offline instance store-backed instances, OpsWorks Stacks automatically installs the new version when the instances are restarted.

Red Hat Enterprise Linux: Supported Node.js Versions

RHEL Version	Node.js Versions
7	(Node.js versions only apply to Chef 11.10 stacks) 0.8.19 0.8.26 0.10.11 0.10.21 0.10.24 0.10.25 0.10.27 0.10.29 0.10.40 0.12.10 0.12.12 0.12.13 0.12.15

Red Hat Enterprise Linux: Supported Chef Versions

Chef Version	Supported RHEL Version
12	Red Hat Enterprise Linux 7
11.10	Red Hat Enterprise Linux 7
11.4 (deprecated)	(None supported)

All versions of Node.js that are older than 0.10.40 are deprecated. 0.12.7 and 0.12.9 are also deprecated.

> **Note**
>
> OpsWorks Stacks supports Apache 2.4 for RHEL 7 instances.

Microsoft Windows Server

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The following notes describe OpsWorks Stacks support for Windows instances. Windows instances are available only for Chef 12.2 stacks. The exact version of Chef in a Windows stack is 12.22.

Currently, the OpsWorks Stacks agent cannot be installed on—and OpsWorks Stacks cannot manage—Windows-based instances that use a system UI language other than **English - United States** (en-US).

Versions

OpsWorks Stacks supports the following Windows 64-bit versions:

- Microsoft Windows Server 2022 Base
- Microsoft Windows Server 2022 with SQL Server Express
- Microsoft Windows Server 2022 with SQL Server Standard
- Microsoft Windows Server 2022 with SQL Server Web
- Microsoft Windows Server 2019 Base
- Microsoft Windows Server 2019 with SQL Server Express
- Microsoft Windows Server 2019 with SQL Server Standard
- Microsoft Windows Server 2019 with SQL Server Web

Creating Instances

You create Windows instances with the OpsWorks Stacks console, API, or CLI. Windows instances are Amazon EBS-backed, but you cannot mount extra Amazon EBS volumes.

Windows stacks can use 24/7 instances, which you start and stop manually. They can also use time-based automatic scaling, which automatically starts and stops instances based on a user-specified schedule. Windows-based stacks cannot use load-based automatic scaling.

You cannot register Windows instances that were created outside of OpsWorks Stacks with a stack.

Updates

AWS updates Windows AMIs for each set of patches, so when you create an instance, it will have the latest updates. However, OpsWorks Stacks does not provide a way to apply updates to online Windows instances. The simplest way to ensure that Windows is up to date is to replace your instances regularly, so that they are always running the latest AMI.

Layers

To handle tasks such as installing and configuring software or deploying apps, you will need to implement one or more custom layers with custom recipes.

Chef

Windows instances use Chef 12.22, and run chef-client in local mode, which launches a local in-memory Chef server called chef-zero. The presence of this server enables custom recipes to use Chef search and data bags.

Remote Login

OpsWorks Stacks provides authorized IAM users with a password that they can use to log in to Windows instances. This password expires after a specified time. Administrators can use an SSH key pair to retrieve an instance's Administrator password, which provides unlimited RDP access. For more information, see Logging In with RDP.

AWS SDK

OpsWorks Stacks automatically installs the AWS SDK for .NET on each instance. This package includes the AWS .NET libraries and AWS Tools for Windows, including the AWS Tools for PowerShell. To use the Ruby SDK, you can have a custom recipe install the appropriate gem.

Monitoring and Metrics

Windows instances support the standard Amazon CloudWatch (CloudWatch) metrics, which you can view in the CloudWatch console.

Ruby

The Chef 12.22 client that OpsWorks Stacks installs on Windows instances comes with Ruby 2.3.6. However, OpsWorks Stacks does not add the executable's directory to the PATH environment variable. To have your applications use this Ruby version, you can typically find it in `C:\opscode\chef\embedded\bin\`.

OpsWorks Stacks Agent CLI

The OpsWorks Stacks agent on Windows instances does not expose a command-line interface.

Proxy Support

Do the following to set up proxy support for Windows instances:

1. Modify `machine.config` to add the following, which adds proxy support to Windows PowerShell (initial bootstrap) and .NET (OpsWorks Stacks agent) applications:

```
<system.net>
  <defaultProxy>
    <proxy autoDetect="false" bypassonlocal="true"
 proxyaddress="http://10.100.1.91:3128"  usesystemdefault="false" />
    <bypasslist>
      <add address="localhost" />
      <add address="169.254.169.254" />
    </bypasslist>
  </defaultProxy>
</system.net>
```

2. Run the following commands to set environment variables for later use by Chef and Git:

```
setx /m no_proxy "localhost,169.254.169.254"
setx /m http_proxy "http://10.100.1.91:3128"
setx /m https_proxy "http://10.100.1.91:3128"
```

> **Note**
>
> For more control over how OpsWorks Stacks updates your instances, create a custom AMI based on Microsoft Windows Server 2022 Base. For example, with custom AMIs you can specify which software is installed on an instance, such as Web Server (IIS). For more information, see Using Custom AMIs.

Adding an Instance to a Layer

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

After you create a layer, you usually add at least one instance. You can add more instances later, if the current set can't handle the load. You can also use load-based or time-based instances to automatically scale the number of instances.

You can add either new or existing instances to a layer:

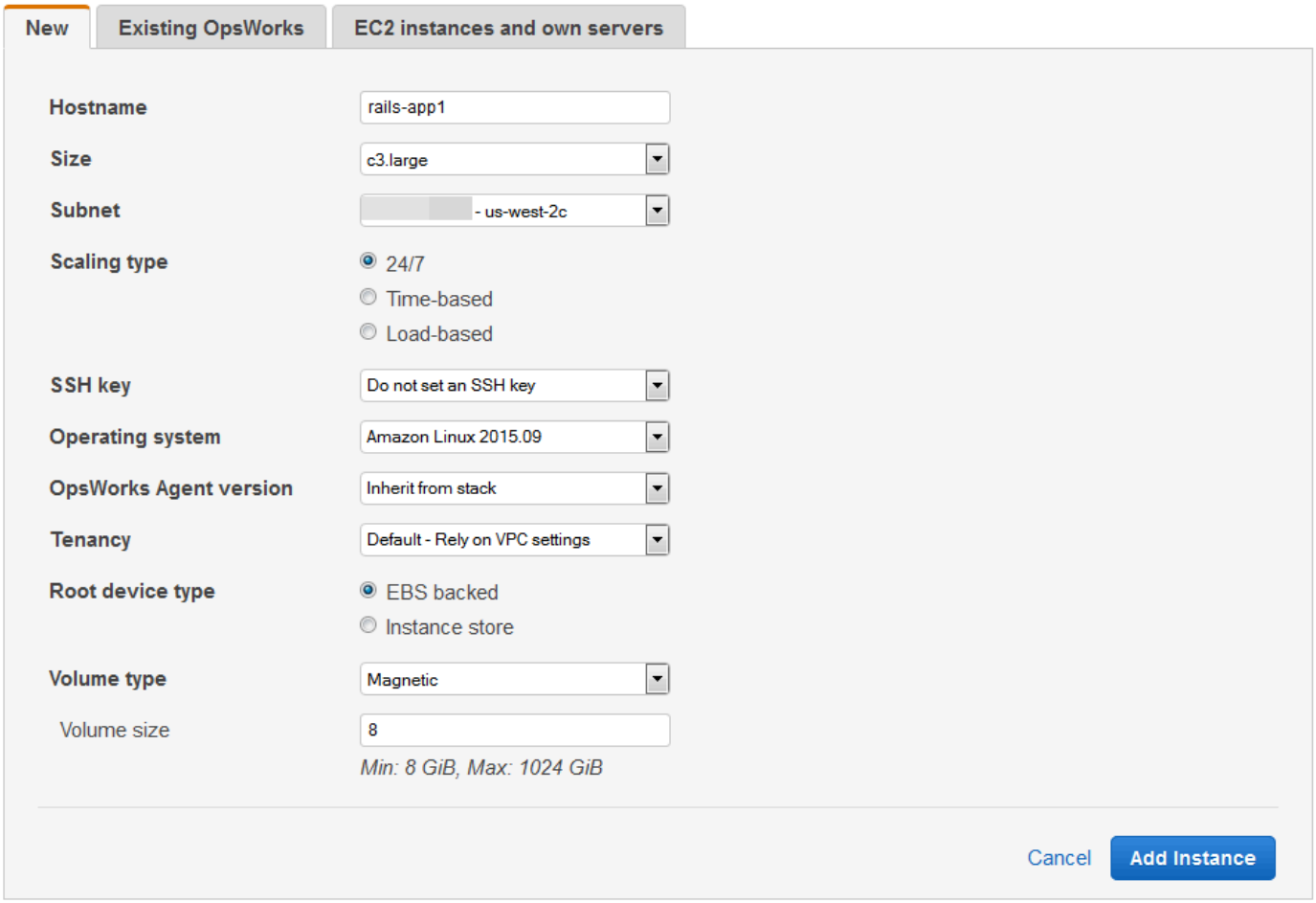
- **New**–OpsWorks creates a new instance, configured to your specifications, and makes it a member of the layer.
- **Existing**–You can add an existing instance from any compatible layer, but it must be in the offline (stopped) state.

If an instance belongs to multiple layers, OpsWorks Stacks runs the recipes for each of the instance's layers when a lifecycle event occurs, or when you run a stack or deployment command.

You can also make an instance a member of multiple layers by editing its configuration. For more information, see Editing the Instance Configuration.

To add a new instance to a layer

1. On the **Instances** page, choose **+Instance** for the appropriate layer and (if necessary) choose the **New** tab. If you want to configure more than just the **Host name**, **Size**, and **Subnet** or **Availability Zone**, choose **Advanced >>** to see more options. The following shows the complete set of options:



2. If desired, you can override the default configuration, most of which you specified when you created the stack. For more information, see Create a New Stack.

   Hostname

   Identifies the instance on the network. By default, OpsWorks Stacks generates each instance's host name by using the **Hostname theme** you specified when you created the stack. You can override this value and specify your preferred host name.

Size

An Amazon EC2 instance type, which specifies the instance's resources, such as the amount of memory or number of virtual cores. OpsWorks Stacks specifies a default size for each instance, which you can override with your preferred instance type.

The instance types supported by OpsWorks Stacks depend on whether or not the stack is in a VPC. Instance types are also limited if your account is using the AWS Free Tier. The drop-down **Size** list shows the supported instance types for the Chef version that your stack supports. Be aware that micro instances such as t1.micro might not have sufficient resources to support some layers. For more information, see Instance Types.

> **Note**
>
> If you are using load-balanced instances, note that Configure lifecycle events can produce a significant CPU load spike that might last a minute or longer. With smaller instances this load spike can be enough to trigger upscaling, especially for large load-balanced stacks with frequent Configure events. The following are some ways to reduce the likelihood of a Configure event causing needless upscaling.
>
> - Use larger instances, so that the additional load from a Configure event is not enough to trigger upscaling.
> - Don't use instance types such as T2 that share CPU resources.
>
> This ensures that when a Configure event occurs, all of the instance's CPU resources are immediately available.
> - Make the `exceeded threshold` time significantly longer than the time required to process a Configure event, perhaps 5 minutes.
>
> For more information, see Using automatic load-based scaling.

Availability Zone/Subnet

If the stack is not in a VPC, this setting is labeled **Availability Zone** and lists the region's zones. You can use this setting to override the default Availability Zone you specified when you created the stack.

If the stack is running in a VPC, this setting is labeled **Subnet** and lists the VPC's subnets. You can use this setting to override the default subnet you specified when you created the stack.

> **Note**
>
> By default, OpsWorks Stacks lists the subnet's CIDR ranges. To make the list more readable, use the VPC console or API to add a tag to each subnet with **Key** set to **`Name`** and **Value** set to the subnet's name. OpsWorks Stacks appends that name to the CIDR range. In the preceding example, the subnet's Name tag is set to **`Private`**.

Scaling Type

Determines how the instance is started and stopped.

- The default value is a **24/7** instance, which you start and stop manually.
- OpsWorks Stacks starts and stops **time-based** instances based on a specified schedule.
- (Linux only) OpsWorks Stacks starts and stops **load-based** instances based on specified load metrics.

> **Note**
>
> You do not start or stop load-based or time-based instances yourself. Instead, you configure the instances, and OpsWorks Stacks starts and stops them based on the configuration. For more information, see Managing load with time-based and load-based instances.

SSH key

An Amazon EC2 key pair. OpsWorks Stacks installs the public key on the instance.

- For Linux instances, you can use the corresponding private key with an SSH client to log in to the instance.
- For Windows instances, you can use the corresponding private key to retrieve the instance's Administrator password. You can then use that password with RDP to log into the instance as Administrator.

Initially, this setting is the **Default SSH key** value that you specified when you created the stack.

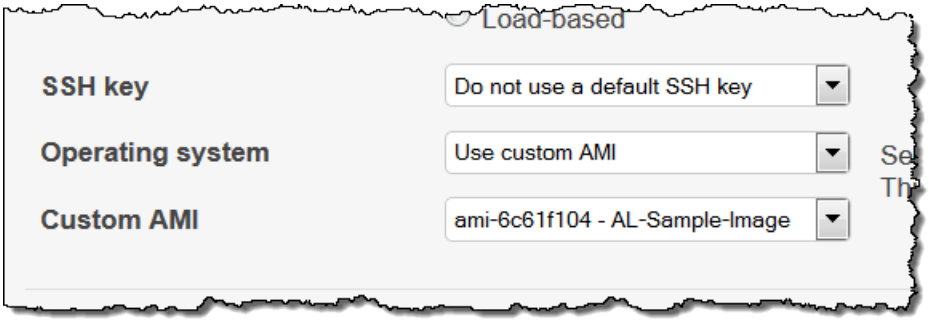
- If the default value is set to **Do not use a default SSH key**, you can specify one of your account's Amazon EC2 keys.
- If the default value is set to an Amazon EC2 key, you can specify a different key or no key.

Operating system

Operating system specifies which operating system the instance is running. OpsWorks Stacks supports only 64-bit operating systems.

Initially, this setting is the **Default operating system** value that you specified when you created the stack. You can override the default value to specify a different Linux operating system or a custom Amazon Machine Image (AMI). However, you cannot switch from Linux to Windows or from Windows to Linux.

If you select **Use custom AMI**, the page displays a list of custom AMIs instead of **Architecture** and **Root device type**.



For more information, see Using Custom AMIs.

OpsWorks Agent version

OpsWorks Agent version specifies the version of the OpsWorks Stacks agent that you want to run on the instance. If you want OpsWorks Stacks to update the agent automatically, choose **Inherit from stack**. To install a specific version of the agent, and manually update the agent on the instance, choose a version from the drop-down list.

> **Note**
>
> Not all agent versions work with all operating system releases. If your instance is running an agent–or you install an agent on an instance–that is not fully supported

on the instance operating system, the OpsWorks Stacks console displays error messages that instruct you to install a compatible agent.

Tenancy

Choose the tenancy option for your instance. You can choose to run your instances on physical servers fully dedicated for your use.

- **Default - Rely on VPC settings**. No tenancy, or inherits tenancy settings from your VPC.
- **Dedicated - Run a dedicated instance**. Pay by the hour for instances that run on single-tenant hardware. For more information, see Dedicated Instances in the Amazon VPC User Guide, and Amazon EC2 Dedicated Instances.
- **Dedicated host - Run this instance on a dedicated host**. Pay for a physical host that is fully dedicated to running your instances, and bring your existing per-socket, per-core, or per-VM software licenses to reduce costs. For more information, see Dedicated Hosts Overview in the Amazon EC2 documentation, and Amazon EC2 Dedicated Hosts.

Root device type

Specifies the instance's root device storage.

- Linux instances can be either Amazon EBS-backed or instance store-backed.
- Windows instances must be Amazon EBS-backed.

For more information, see Storage.

> **Note**
>
> After the initial boot, Amazon EBS-backed instances boot faster than instance store-backed instances because OpsWorks Stacks does not have to reinstall the instance's software from scratch. For more information, see Root Device Storage.

Volume type

Specifies the root device volume type: **Magnetic**, **Provisioned IOPS (SSD)**, or **General Purpose (SSD)**. For more information, see Amazon EBS Volume Types.

Volume size

Specifies the root device volume size for the specified volume type. For more information, see Amazon EBS Volume Types.

- **General Purpose (SSD)**. Minimum allowed size is: 8 GiB; maximum size is 16384 GiB.
- **Provisioned IOPS (SSD)**. Minimum allowed size is: 8 GiB; maximum size is 16384 GiB. You can set a minimum of 100 input/output operations per second (IOPS), and a maximum of 240 IOPS.
- **Magnetic**. Minimum allowed size is 8 GiB; maximum size is 1024 GiB.

3. Choose **Add Instance** to create the new instance.

> **Note**
>
> You cannot override the stack's default agent version setting when you create an instance. To specify a custom agent version setting, you must create the instance and then edit its configuration.

To add an existing instance to a layer

1. On the **Instances** page, choose **+Instance** for the appropriate layer, and then open the **Existing** tab.

   > **Note**
   >
   > If you change your mind about using an existing instance, choose **New** to create a new instance as described in the preceding procedure.

2. On the **Existing** tab, select an instance from the list.
3. Choose **Add Instance** to create the new instance.

An instance represents an Amazon EC2 instance, but is basically just an OpsWorks Stacks data structure. An instance must be started to create a running Amazon EC2 instance, as described in the following sections.

> **Important**
>
> If you launch instances into a default VPC, you must be careful about modifying the VPC configuration. The instances must always be able to communicate with the OpsWorks Stacks service, Amazon S3, and package repositories. If, for example, you remove a default gateway, the instances will lose their connection to the OpsWorks Stacks service, which will then treat the instances as failed and auto heal them. However, OpsWorks Stacks will not be able to install the instance agent on the healed instances. Without an agent, the instances cannot communicate with the service, and the startup process will not progress beyond the `booting` status. For more information on default VPC, see Supported Platforms.

You can also incorporate Linux computing resources into a stack that were created outside of OpsWorks Stacks:

- Amazon EC2 instances that you created directly by using the Amazon EC2 console, CLI, or API.
- *On-premises* instances running on your own hardware, including instances running in virtual machines.

For more information, see Using Computing Resources Created Outside of OpsWorks Stacks.

Using Custom AMIs

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks supports two ways to customize instances: custom Amazon Machine Images (AMIs) and Chef recipes. Both approaches give you control over which packages and package versions are installed, how they are configured, and so on. However, each approach has different advantages, so the best one depends on your requirements.

The following are the primary reasons to consider using a custom AMI:

- You want to prebundle specific packages instead of installing them after the instance boots.
- You want to control the timing of package updates to provide a consistent base image for your layer.
- You want instances—load-based instances in particular—to boot as quickly as possible.

The following are the primary reasons to consider using Chef recipes:

- They are more flexible than custom AMIs.
- They are easier to update.
- They can perform updates on running instances.

In practice, the optimal solution might be a combination of both approaches. For more information about recipes, see Cookbooks and Recipes.

Topics

- How Custom AMIs work with OpsWorks Stacks
- Creating a Custom AMI for OpsWorks Stacks

How Custom AMIs work with OpsWorks Stacks

To specify a custom AMI for your instances, select **Use custom AMI** as the instance's operating system when you create a new instance. OpsWorks Stacks then displays a list of the custom AMIs in the stack's region and you select the appropriate one from the list. For more information, see Adding an Instance to a Layer.

> **Note**
>
> You cannot specify a particular custom AMI as a stack's default operating system. You can set `Use custom AMI` as the stack's default operating system, but you can specify a particular AMI only when you add new instances to layers. For more information, see Adding an Instance to a Layer and Create a New Stack. While it might be possible to create instances with other operating systems (such as CentOS 6.*x*) that have been created from custom or community-generated AMIs, these are not officially supported.

This topic discusses some general issues that you should consider before creating or using a custom AMI.

Topics

- Startup Behavior
- Choosing a Layer
- Handling Applications

Startup Behavior

When you start the instance, OpsWorks Stacks uses the specified custom AMI to launch a new Amazon EC2 instance. OpsWorks Stacks then uses cloud-init to install the OpsWorks Stacks agent on the instance and the agent runs the instance's Setup recipes followed by the Deploy recipes. After the instance is online, the agent runs the Configure recipes for every instance in the stack, including the newly added instance.

Choosing a Layer

The OpsWorks Stacks agent usually does not conflict with installed packages. However, the instance must be a member of at least one layer. OpsWorks Stacks always runs that layer's recipes, which could cause problems. You should understand exactly what a layer's recipes do to an instance before adding an instance with a custom AMI to that layer.

To see which recipes a particular layer type runs on your instance, open a stack that includes that layer. Then click **Layers** in the navigation pane, and click **Recipes** for the layer of interest. To see the actual code, click the recipe name.

> **Note**
>
> For Linux AMIs, one way to reduce the possibility of conflicts is to use OpsWorks Stacks to provision and configure the instance that is the basis for your custom AMI. For more information, see Create a Custom Linux AMI from an OpsWorks Stacks Instance.

Handling Applications

In addition to packages, you might also want to include an application in the AMI. If you have a large complex application, including it in the AMI can shorten the instance's startup time. You can

include small applications in your AMI, but there is usually little or no time advantage relative to having OpsWorks Stacks deploy the application.

One option is to include the application in your AMI and also create an app that deploys the application to the instances from a repository. This approach shortens your boot time but also provides a convenient way to update the application after the instance is running. Note that Chef recipes are idempotent, so the deployment recipes won't modify the application as long as the version in the repository is the same as the one on the instance.

Creating a Custom AMI for OpsWorks Stacks

To use a custom AMI with OpsWorks Stacks, you must first create an AMI from a customized instance. You can choose from two options:

- Use the Amazon EC2 console or API to create and customize an instance, based on a 64-bit version of one of the OpsWorks Stacks-supported AMIs.
- For Linux AMIs, use OpsWorks to create an Amazon EC2 instance, based on the configuration of its associated layers.

Before you create a custom Linux AMI, disable `noexec` on the `/tmp` partition to allow OpsWorks Stacks to install its agent on custom Linux instances.

> **Note**
>
> Be aware that an AMI might not work with all instance types, so make sure that your starting AMI is compatible with the instance types that you plan to use. In particular, the R3 instance types require a hardware-assisted virtualization (HVM) AMI.

You then use the Amazon EC2 console or API to create a custom AMI from the customized instance. You can use your custom AMIs in any stack that is in the same region by adding an instance to a layer and specifying your custom AMI. For more information on how to create an instance that uses a custom AMI, see Adding an Instance to a Layer.

> **Note**
>
> By default, OpsWorks Stacks installs all Amazon Linux updates on boot, which provides you with the latest release. In addition, Amazon Linux releases a new version approximately

every six months, which can involve significant changes. By default, custom AMIs based on Amazon Linux are automatically updated to the new version when it is released. The recommended practice is to lock your custom AMI to a specific Amazon Linux version, which allows you to defer the update until you have tested the new version. For more information, see How do I lock my AMI to a specific version?.

Topics

- Create a Custom AMI using Amazon EC2
- Create a Custom Linux AMI from an OpsWorks Stacks Instance
- Create a Custom Windows AMI

Create a Custom AMI using Amazon EC2

The simplest way to create a custom AMI—and the only option for Windows AMIs—is to perform the entire task by using the Amazon EC2 console or API. For more details about the following steps, see Creating Your Own AMIs.

To create a custom AMI using Amazon EC2 console or API

1. Create an instance by using a 64-bit version of one of the OpsWorks Stacks-supported AMIs.
2. Customize the instance from Step 1 by configuring it, installing packages, and so on. Remember that everything you install will be reproduced on every instance based on the AMI, so don't include items that should be specific to a particular instance.
3. Stop the instance and create a custom AMI.

Create a Custom Linux AMI from an OpsWorks Stacks Instance

To use a customized OpsWorks Stacks Linux instance to create an AMI, be aware that every Amazon EC2 instance created by OpsWorks includes a unique identity. If you create a custom AMI from such an instance, it includes that identity, and all instances based on the AMI have the same identity. To ensure that the instances based on your custom AMI have a unique identity, you must remove the identity from the customized instance before creating the AMI.

To create a custom AMI from an OpsWorks Stacks instance

1. Create a Linux stack and add one or more layers to define the configuration of the customized instance. You can use built-in layers, customized as appropriate, as well as fully custom layers. For more information, see Customizing OpsWorks Stacks.
2. Edit the layers and disable AutoHealing.
3. Add an instance with your preferred Linux distribution to the layer or layers and start it. We recommend using an Amazon EBS-backed instance. Open the instance's details page and record its Amazon EC2 ID for later.
4. When the instance is online, log in with SSH, and perform one of the next four steps, depending upon your instance operating system.
5. For an Amazon Linux instance in either a Chef 11 or Chef 12 stack, or a Red Hat Enterprise Linux 7 instance in a Chef 11 stack, do the following.

   a. `sudo /etc/init.d/monit stop`

   b. `sudo /etc/init.d/opsworks-agent stop`

   c. `sudo rm -rf /etc/aws/opsworks/ /opt/aws/opsworks/ /var/log/aws/opsworks/ /var/lib/aws/opsworks/ /etc/monit.d/opsworks-agent.monitrc /etc/monit/conf.d/opsworks-agent.monitrc /var/lib/cloud/ /etc/chef`

   > **Note**
   >
   > For instances in a Chef 12 stack, add the following two folders to this command:
   >
   > - `/var/chef`
   > - `/opt/chef`

   d. `sudo rpm -e opsworks-agent-ruby`

   e. `sudo rpm -e chef`

6. For an Ubuntu 16.04 LTS or 18.04 LTS instance in a Chef 12 stack, do the following.

   a. `sudo systemctl stop opsworks-agent`

   b. `sudo rm -rf /etc/aws/opsworks/ /opt/aws/opsworks/ /var/log/aws/opsworks/ /var/lib/aws/opsworks/ /etc/monit.d/opsworks-`

```
agent.monitrc /etc/monit/conf.d/opsworks-agent.monitrc /var/lib/
cloud/ /var/chef /opt/chef /etc/chef
```

c. `sudo apt-get -y remove chef`

d. `sudo dpkg -r opsworks-agent-ruby`

e. `systemctl stop apt-daily.timer`

f. `systemctl stop apt-daily-upgrade.timer`

g. `rm /var/lib/systemd/timers/stamp-apt-daily.timer`

h. `rm /var/lib/systemd/timers/stamp-apt-daily-upgrade.timer`

7. For other supported Ubuntu versions in a Chef 12 stack, do the following.

a. `sudo /etc/init.d/monit stop`

b. `sudo /etc/init.d/opsworks-agent stop`

c. `sudo rm -rf /etc/aws/opsworks/ /opt/aws/opsworks/ /var/log/aws/opsworks/ /var/lib/aws/opsworks/ /etc/monit.d/opsworks-agent.monitrc /etc/monit/conf.d/opsworks-agent.monitrc /var/lib/cloud/ /var/chef /opt/chef /etc/chef`

d. `sudo apt-get -y remove chef`

e. `sudo dpkg -r opsworks-agent-ruby`

8. For a Red Hat Enterprise Linux 7 instance in a Chef 12 stack, do the following.

a. `sudo systemctl stop opsworks-agent`

b. `sudo rm -rf /etc/aws/opsworks/ /opt/aws/opsworks/ /var/log/aws/opsworks/ /var/lib/aws/opsworks/ /etc/monit.d/opsworks-agent.monitrc /etc/monit/conf.d/opsworks-agent.monitrc /var/lib/cloud/ /etc/chef /var/chef`

c. `sudo rpm -e opsworks-agent-ruby`

d. `sudo rpm -e chef`

9. This step depends on the instance type:

- For an Amazon EBS-backed instance, use the OpsWorks Stacks console to stop the instance and create the AMI as described in Creating an Amazon EBS-Backed Linux AMI..
- For an instance store-backed instance, create the AMI as described in Creating an Instance Store-Backed Linux AMI and then use the OpsWorks Stacks console to stop the instance.

When you create the AMI, be sure to include the certificate files. For example, you can call the `ec2-bundle-vol` command with the `-i` argument set to `-i $(find /etc /usr /opt -name '*.pem' -o -name '*.crt' -o -name '*.gpg' | tr '\n' ',')`. Do not remove the apt public keys when bundling. The default `ec2-bundle-vol` command handles this task.

10. Clean up your stack by returning to the OpsWorks Stacks console and deleting the instance from the stack.

Create a Custom Windows AMI

The following procedures create custom AMIs for Windows Server 2022 Base. You can choose other Windows Server operating systems in the Amazon EC2 management console.

> **Important**
>
> Currently, the OpsWorks Stacks agent cannot be installed on—and OpsWorks Stacks cannot manage—Windows-based instances that use a system UI language other than **English - United States** (en-US).

Topics

- Creating a Custom Windows AMI with Sysprep
- Creating a Custom Windows AMI Without Sysprep
- Adding a New Instance by Using a Custom Windows AMI

Creating a Custom Windows AMI with `Sysprep`

Creating custom Windows AMIs by using Sysprep typically results in a slower instance launch, but a cleaner process. The first-time startup of an instance created from an image created with `Sysprep` takes more time because of `Sysprep` activities, restarts, OpsWorks Stacks provisioning, and the first OpsWorks Stacks run, including setup and configuration. Complete the steps for creating a custom Windows AMI in the Amazon EC2 console.

To create a custom Windows AMI with Sysprep

1. In the Amazon EC2 console, choose **Launch Instance**.

2. Find **Microsoft Windows Server 2022 Base**, and then choose **Select**.
3. Choose the instance type that you want, and then choose **Configure Instance Details**. Make configuration changes to the AMI, including machine name, storage, and security group settings. Choose **Launch**.
4. After the instance boot process finishes, get your password, and then connect to the instance in a Windows **Remote Desktop Connection** window.
5. On the Windows **Start** screen, choose **Start**, and then begin typing **`ec2configservice`** until the results show the **EC2ConfigServiceSettings** console. Open the console.
6. On the **General** tab, make sure that the **Enable UserData execution** check box is filled (although this option is not required for `Sysprep`, it is required for OpsWorks Stacks to install its agent). Clear the check box for the **Set the computer name of the instance...** option, because this option can cause a restart loop with OpsWorks Stacks.
7. On the **Image** tab, set **Administrator Password** to either **Random** to allow Amazon EC2 to automatically generate a password that you can retrieve with an SSH key, or **Specify** to specify your own password. `Sysprep` saves this setting. If you specify your own password, store the password in a convenient place. We recommend that you do not choose **Keep Existing**.
8. Choose **Apply**, and then choose **Shutdown with Sysprep**. When you are prompted to confirm, choose **Yes**.
9. After the instance has stopped, in the Amazon EC2 console, right-click the instance in the **Instances** list, choose **Image**, and then choose **Create Image**.
10. On the **Create Image** page, provide a name and description for the image, and specify the volume configuration. When you have finished, choose **Create Image**.
11. Open the **Images** page, and wait for your image to change from the **pending** stage to **available**. Your new AMI is ready to use.

Creating a Custom Windows AMI Without `Sysprep`

Complete the steps for creating a custom Windows AMI in the Amazon EC2 console.

To create a custom Windows AMI without Sysprep

1. In the Amazon EC2 console, choose **Launch Instance**.
2. Find **Microsoft Windows Server 2022 Base**, and then choose **Select**.

3. Choose the instance type that you want, and then choose **Configure Instance Details**. Make configuration changes to the AMI, including machine name, storage, and security group settings. Choose **Launch**.
4. After the instance boot process finishes, get your password, and then connect to the instance in a Windows **Remote Desktop Connection** window.
5. On the instance, open `C:\Program Files\Amazon\Ec2ConfigService\Settings\config.xml`, change the following two settings, and then save and close the file:
   - `Ec2SetPassword` to `Enabled`
   - `Ec2HandleUserData` to `Enabled`
6. Disconnect from the **Remote Desktop** session, and return to the Amazon EC2 console.
7. In the **Instances** list, stop the instance.
8. After the instance has stopped, in the Amazon EC2 console, right-click the instance in the **Instances** list, choose **Image**, and then choose **Create Image**.
9. On the **Create Image** page, provide a name and description for the image, and specify the volume configuration. When you have finished, choose **Create Image**.
10. Open the **Images** page, and wait for your image to change from the **pending** stage to **available**. Your new AMI is ready to use.

Adding a New Instance by Using a Custom Windows AMI

After your image changes to the **available** state, you can create new instances that are based on your custom Windows AMI. When you choose **Use custom Windows AMI** from the **Operating system** list, OpsWorks Stacks displays a list of custom AMIs.

To add a new instance based on a custom Windows AMI

1. When your new AMI is available, go to the OpsWorks Stacks console, open the **Instances** page for a Windows stack, and choose **+ Instance** near the bottom of the page to add a new instance.
2. On the **New** tab, choose **Advanced**.
3. On the **Operating system** drop-down list, choose **Use custom Windows AMI**.
4. On the **Custom AMI** drop-down list, choose the AMI that you created, and then choose **Add Instance**.

You can now start and run the instance.

Manually Starting, Stopping, and Rebooting 24/7 Instances

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

You can use 24/7 instances with both Linux and Windows stacks.

After you add a 24/7 instance to a layer, you must manually start the instance to launch the corresponding Amazon Elastic Compute Cloud (Amazon EC2) instance and manually stop it to terminate the Amazon EC2 instance. You can also manually reboot instances that are not functioning properly. OpsWorks Stacks automatically starts and stops time-based and load-based instances. For more information, see Managing load with time-based and load-based instances.

Important

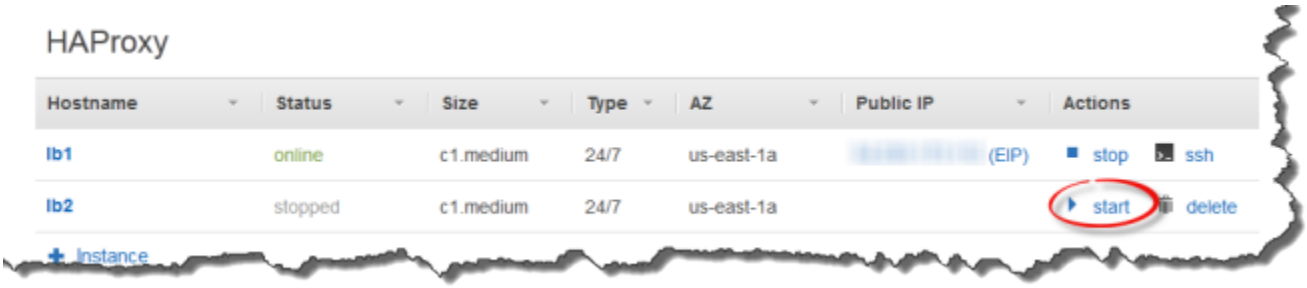
OpsWorks Stacks instances must be started, stopped, and restarted only in the OpsWorks console. OpsWorks doesn't recognize start, stop, or restart operations performed in the Amazon EC2 console.

Topics

- Starting or Restarting an Instance
- Stopping an Instance
- Rebooting an Instance

Starting or Restarting an Instance

To start a new instance, on the **Instances** page, click **start** in the instance's **Actions** column.



You can also create multiple instances and then start them all at the same time by clicking **Start all Instances**.

After you start the instance, OpsWorks Stacks launches an Amazon EC2 instance and boots the operating system. The startup process usually takes a few minutes, and is typically somewhat slower for Windows instances than for Linux instances. As startup progresses, the instance's **Status** field displays the following series of values:

1. **requested** - OpsWorks Stacks has called the Amazon EC2 service to create the Amazon EC2 instance.
2. **pending** - OpsWorks Stacks is waiting for the Amazon EC2 instance to start.
3. **booting** - The Amazon EC2 instance is booting.
4. **running_setup** - OpsWorks Stacks has triggered the Setup event and is running the layer's Setup recipes, followed by its Deploy recipes. For more information, see Executing Recipes. If you have added custom cookbooks to the stack, OpsWorks Stacks installs the current version from your repository before running the Setup and Deploy recipes.
5. **online** - The instance is ready for use.

When the **Status** changes to **online**, the instance is fully operational.

- If the layer has an attached load balancer, OpsWorks Stacks adds the instance to it.
- OpsWorks Stacks triggers a Configure event, which runs each instance's Configure recipes.

  As needed, these recipes update the instance to accommodate the new instance.
- OpsWorks Stacks replaces the instance's **start** action with **stop**, which you can use to stop the instance.

If the instance did not start successfully or the setup recipes failed, the status will be set to **start_failed** or **setup_failed**, respectively. You can examine the logs to determine the cause. For more information, see Debugging and Troubleshooting Guide.

A stopped instance remains part of the stack and retains all resources. For example, Amazon EBS volumes and Elastic IP addresses are still associated with a stopped instance. You can restart a stopped instance by choosing **start** in the instance's **Actions** column. Restarting a stopped instance does the following:

- Instance store-backed instances – OpsWorks Stacks launches a new Amazon EC2 instance with the same configuration.
- Amazon EBS-backed instances – OpsWorks Stacks restarts the Amazon EC2 instance, which reattaches the root volume.

After the instance finishes booting, OpsWorks Stacks installs operating system updates and runs the Setup and Deploy recipes, just as with the initial start. OpsWorks Stacks also does the following for restarted instances, as appropriate.

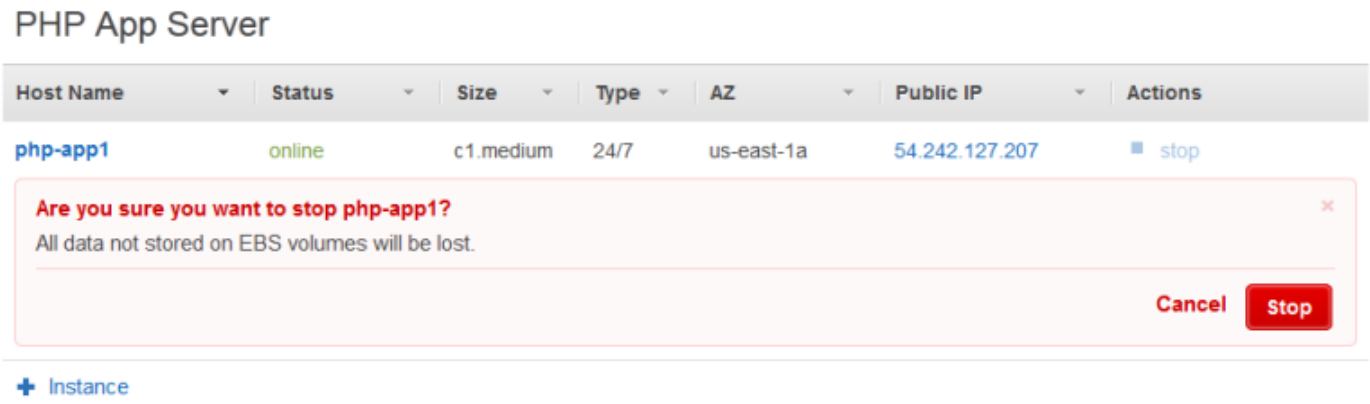
- Reassociates Elastic IP addresses.
- Reattaches Amazon Elastic Block Store (Amazon EBS) volumes.
- For instance store-backed instances, installs the latest cookbook versions.

  Amazon EBS-backed instances continue to use the custom cookbooks that were stored on the root volume. If your custom cookbooks have changed since you stopped the instance, you must manually update them after the instance is online. For more information, see Updating Custom Cookbooks.

> **Note**
>
> It might take several minutes for an Elastic IP address to be reassociated with a restarted instance. Be aware that the instance's **Elastic IP** setting represents metadata, and simply indicates that the address should be associated with the instance. The **Public IP** setting reflects the instance's state, and might be empty initially. When the Elastic IP address is associated with the instance, the address is assigned to the **Public IP** setting, followed by (EIP).

Stopping an Instance

On the **Instances** page, click **stop** in the instance's **Actions** column, which notifies OpsWorks Stacks to run the shutdown recipes and terminate the EC2 instance.



You can also shut down every instance in the stack by clicking **Stop All Instances**.

After you stop the instance, OpsWorks Stacks performs several tasks:

1. If the instance's layer has an attached Elastic Load Balancing load balancer, OpsWorks Stacks deregisters the instance.

   If the layer supports the load balancer's connection draining feature, OpsWorks Stacks delays triggering the Shutdown event until connection draining is complete. For more information, see Elastic Load Balancing Layer.
2. OpsWorks Stacks triggers a Shutdown event, which runs the instance's Shutdown recipes.
3. After triggering the Shutdown event, OpsWorks Stacks waits for a specified time to allow the Shutdown recipes time to finish and then does the following:
   - Terminates instance store-backed instances, which deletes all data.
   - Stops Amazon EBS-backed instances, which preserves the data on the root volume.

   For more information on instance storage, see Storage.

> **Note**
>
> The default shutdown timeout setting is 120 seconds. If your Shutdown recipes need more time, you can edit the layer configuration to change the setting.

You can monitor the shutdown process by watching the instance's **Status** column. As shutdown progresses, it displays the following series of values:

1. **terminating** - OpsWorks Stacks is terminating the Amazon EC2 instance.
2. **shutting_down** - OpsWorks Stacks is running the layer's Shutdown recipes.
3. **terminated** - The Amazon EC2 instance is terminated.
4. **stopped** - The instance has stopped.

Rebooting an Instance

On the **Instances** page, click the nonfunctioning instance's name to open the details page and then click **Reboot**.



This command performs a soft reboot of the associated Amazon EC2 instance. It does not delete the instance's data, even for instance store-backed instances, and does not trigger any lifecycle events.

> **Note**
>
> To have OpsWorks Stacks automatically replace failed instances, enable auto healing. For more information, see Using Auto Healing.

Managing load with time-based and load-based instances

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

As your incoming traffic varies, your stack may have either too few instances to comfortably handle the load or more instances than necessary. You can save both time and money by using time-based or load-based instances to automatically increase or decrease a layer's instances so that you always have enough instances to adequately handle incoming traffic without paying for unneeded capacity. There's no need to monitor server loads or manually start or stop instances. In addition, time- and load-based instances automatically distribute, scale, and balance applications over multiple Availability Zones within a region, giving you geographic redundancy and scalability.

Automatic scaling is based on two instance types, which adjust a layer's online instances based on different criteria:

- **Time-based** instances

  They allow a stack to handle loads that follow a predictable pattern by including instances that run only at certain times or on certain days. For example, you could start some instances after 6PM to perform nightly backup tasks or stop some instances on weekends when traffic is lower.

- **Load-based** instances

  They allow a stack to handle variable loads by starting additional instances when traffic is high and stopping instances when traffic is low, based on any of several load metrics. For example, you can have OpsWorks Stacks start instances when the average CPU utilization exceeds 80% and stop instances when the average CPU load falls below 60%.

Both time-based and load-based instances are supported for Linux stacks, while only time-based instances are supported for Windows stacks.

Unlike 24/7 instances, which you must start and stop manually, you do not start or stop time-based or load-based instances yourself. Instead, you configure the instances and OpsWorks Stacks starts or stops them based on their configuration. For example, you configure time-based instances to start and stop on a specified schedule. OpsWorks Stacks then starts and stops the instances according to that configuration.

A common practice is to use all three instance types together, as follows.

- A set 24/7 instances to handle the base load. You typically just start these instances and let them run continuously.

- A set of time-based instances, which OpsWorks Stacks starts and stops to handle predictable traffic variations. For example, if your traffic is highest during working hours, you would configure the time-based instances to start in the morning and shut down in the evening.
- A set of load-based instances, which OpsWorks Stacks starts and stops to handle unpredictable traffic variations. OpsWorks Stacks starts them when the load approaches the capacity of the stacks' 24/7 and time-based instances, and stops them when the traffic returns to normal..

For more information on how to use these scaling times, see Optimizing the Number of Servers.

Note

If you have created apps for the instances' layer or created custom cookbooks, OpsWorks Stacks automatically deploys the latest version to time-based and load-based instances when they are first started. However, OpsWorks Stacks does not necessarily deploy the latest cookbooks to restarted offline instances. For more information, see Editing Apps and Updating Custom Cookbooks.

Topics

- Using automatic time-based scaling
- Using automatic load-based scaling
- How load-based scaling differs from auto healing

Using automatic time-based scaling

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Time-based scaling lets you control how many instances a layer should have online at certain times of day or days of the week by starting or stopping instances on a specified schedule. OpsWorks

Stacks checks every couple of minutes and starts or stops instances as required. You specify the schedule separately for each instance, as follows:

- Time of day. You can have more instances running during the day than at night, for example.
- Day of the week. You can have more instances running on weekdays than weekends, for example.

> **Note**
>
> You cannot specify particular dates.

Topics

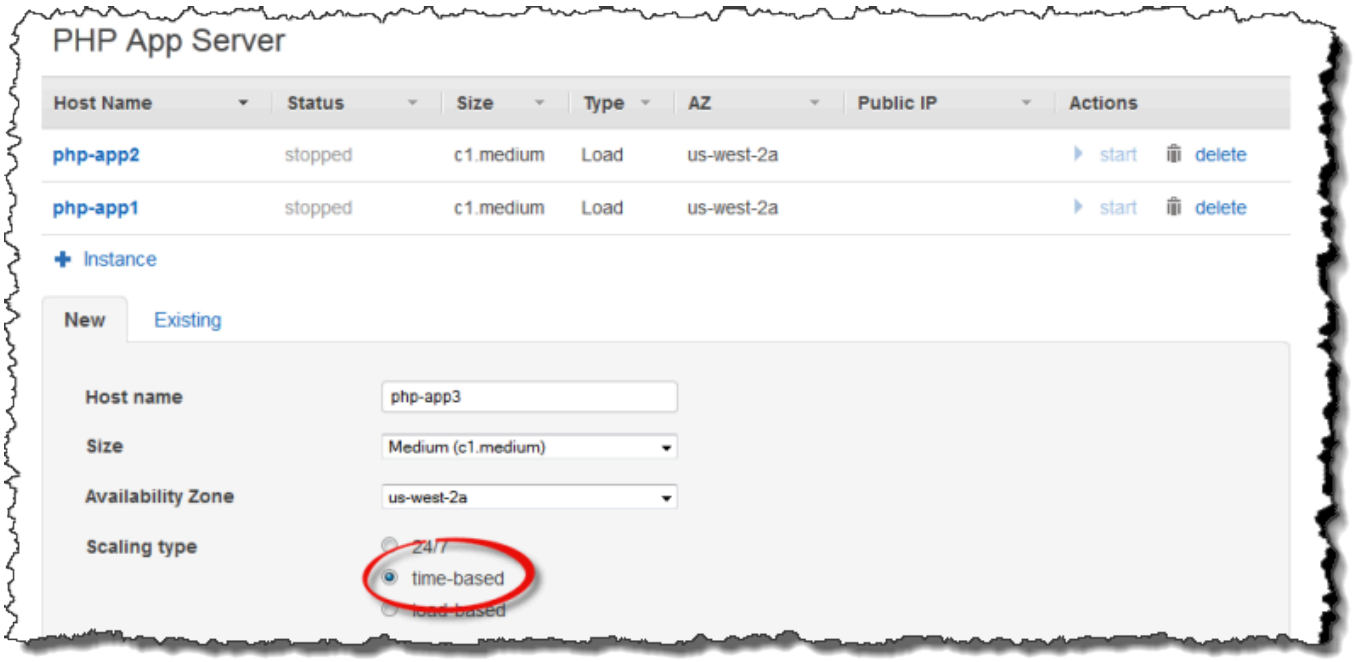
- Adding a time-based instance to a layer
- Configuring a time-based instance

Adding a time-based instance to a layer

You can either add a new time-based instance to a layer, or use an existing instance.

To add a new time-based instance

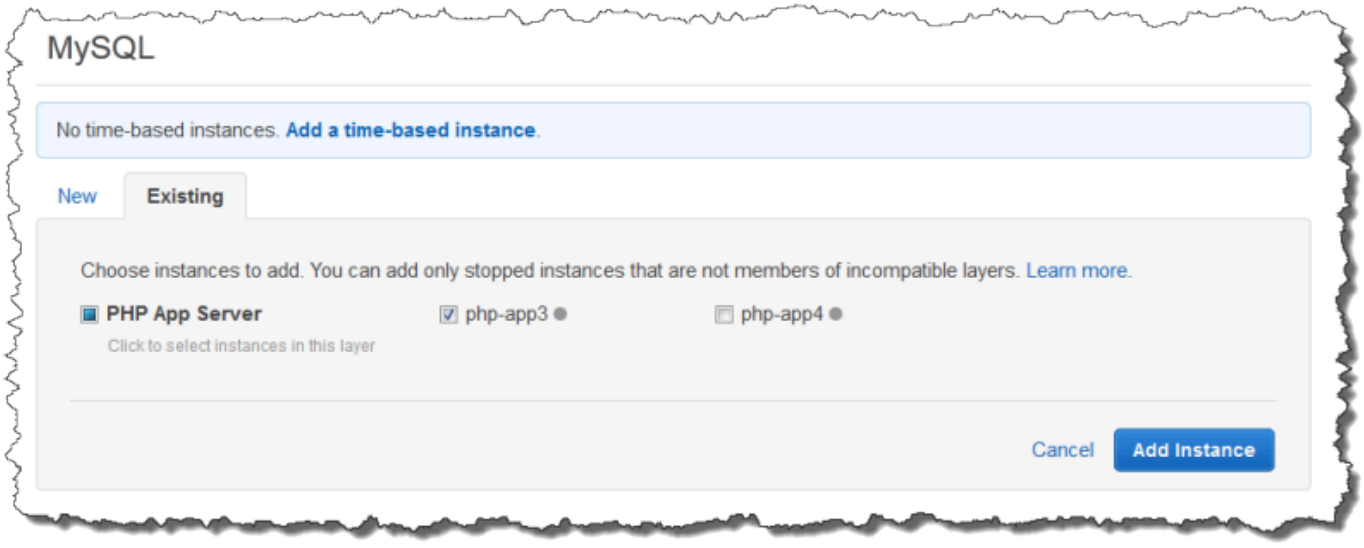
1. On the **Instances** page, choose **+ Instance** to add an instance. On the **New** tab, choose **Advanced**, and then choose **time-based**.



2. Configure the instance. Then choose **Add Instance** to add the instance to the layer.

To add an existing time-based instance to a layer

1. On the **Time-based Instances** page, choose **+ Instance** if a layer already has a time-based instance. Otherwise, choose **Add a time-based instance**. Then choose the **Existing** tab.



2. On the **Existing** tab, choose an instance from the list. The list shows only time-based instances.

> **Note**
>
> If you change your mind about using an existing instance, on the **New** tab, create a new instance, as described in the preceding procedure.

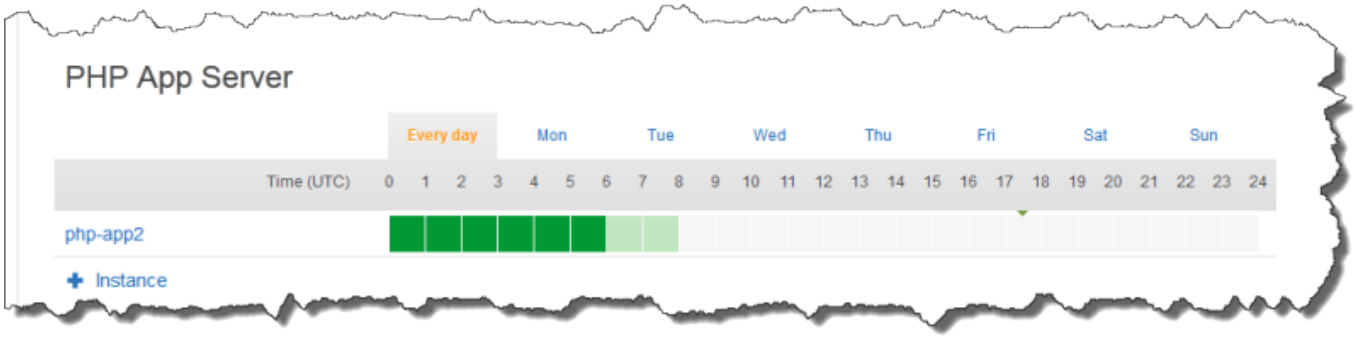
3. Choose **Add instance** to add the instance to the layer.

Configuring a time-based instance

After you add a time-based instance to a layer, you configure its schedule as follows.

To configure a time-based instance

1. In the navigation pane, under **Instances**, choose **Time-based**.
2. Specify the online periods for each time-based instance by filling the appropriate boxes below the desired hour.
   - To use the same schedule every day, choose the **Every day** tab, and then specify the online time periods.
   - To use different schedules on different days, choose each day, and then choose the appropriate time periods.



> **Note**
>
> Be sure to allow for the amount of time it takes to start an instance, and that OpsWorks Stacks checks only every few minutes to see if instances should be started or stopped. For example, if an instance should be running by 1:00 UTC, start it at 0:00 UTC. Otherwise,

OpsWorks Stacks might not start the instance until several minutes past 1:00 UTC, and the instance takes several more minutes to be online.

You can change an instance's online time periods at any time by performing the preceding steps. The next time OpsWorks Stacks checks, it uses the new schedule to determine whether to start or stop instances.

Note

You can add a new time-based instance to a layer by opening the **Time-based** page, and choosing **Add a time-based instance** (if you have not yet added a time-based instance to the layer) or **+ Instance** (if the layer already has one or more time-based instances). Then, configure the instance as described in the preceding procedures.

Using automatic load-based scaling

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Load-based instances let you rapidly start or stop instances in response to changes in incoming traffic. OpsWorks Stacks uses Amazon CloudWatch data to compute the following metrics for each layer, which represent average values across all of the layer's instances:

- CPU: The average CPU consumption, such as 80%
- Memory: The average memory consumption, such as 60%
- Load: The average computational work a system performs in one minute.

You define *upscaling* and *downscaling* thresholds for any or all of these metrics. You can also use custom CloudWatch alarms as thresholds.

Crossing a threshold triggers a *scaling event*. You determine how OpsWorks Stacks responds to scaling events by specifying the following:

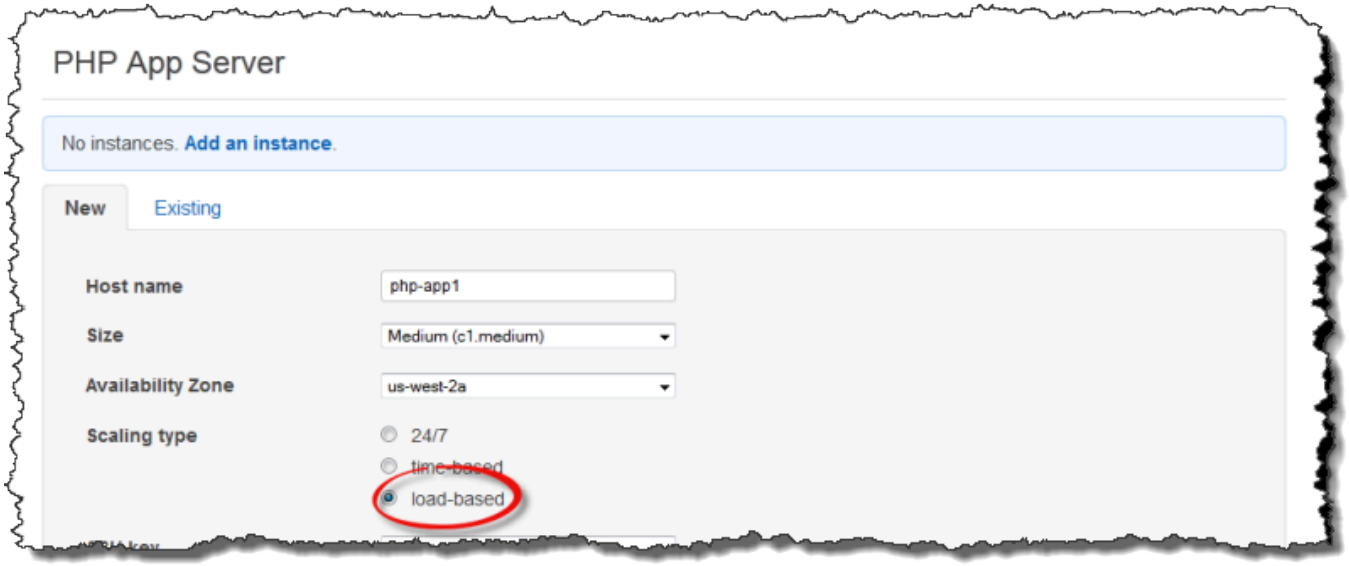
- How many instances to start or stop.
- How long OpsWorks Stacks should wait after exceeding a threshold before starting or deleting instances. For example, CPU utilization must exceed the threshold for at least 15 minutes. This value allows you to ignore brief traffic fluctuations.
- How long OpsWorks Stacks should wait after starting or stopping instances before monitoring metrics again. You usually want to allow enough time for started instances to come online or stopped instances to shut down before assessing whether the layer is still exceeding a threshold.

When a scaling event occurs, OpsWorks Stacks starts or stops only load-based instances. It does not start or stop 24/7 instances or time-based instances.

> **Note**
>
> Automatic load-based scaling does not create new instances; it starts and stops only those instances that you have created. You must therefore provision enough load-based instances in advance to handle the maximum anticipated load.

To create a load-based instance

1. On the **Instances** page, choose **+Instance** to add an instance. Choose **Advanced**, and then choose **load-based**.



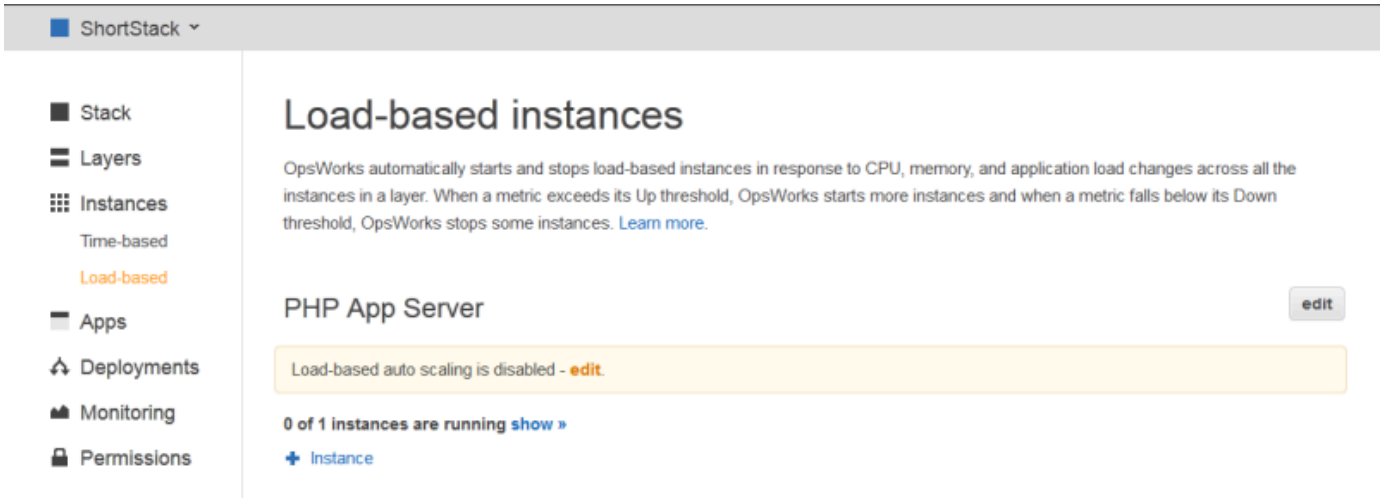
2. Configure the instance, then choose **Add Instance** to add the instance to the layer.

Repeat this procedure until you have created a sufficient number of instances. You can add or remove instances later, as required.

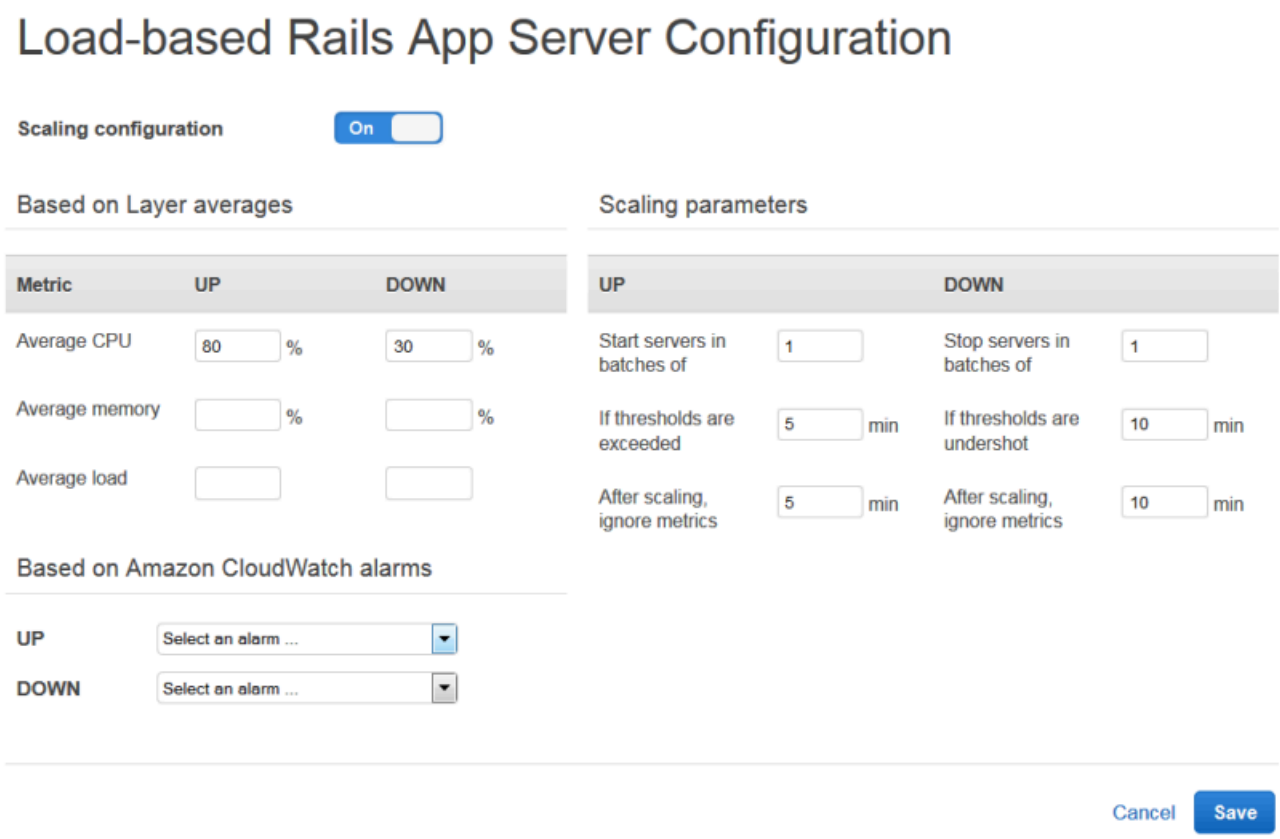
After you have added load-based instances to a layer, you must enable load-based scaling and specify the configuration. The load-based scaling configuration is a layer property, not an instance property, that specifies when a layer should start or stop its load-based instances. It must be specified separately for each layer that uses load-based instances.

To enable and configure automatic load-based scaling

1. In the navigation pane, under **Instances**, choose **Load-based**, and then choose **edit** for the appropriate layer.



2. Set **Load-based auto scaling enabled** to **On**. Then set threshold and scaling parameters to define how and when to add or delete instances.



Layer-average thresholds

You can set scaling thresholds based on the following values, which are averaged over all of the layer's instances.

- **Average CPU** – The layer's average CPU utilization, as a percent of the total.
- **Average memory** – The layer's average memory utilization, as a percent of the total.
- **Average load** – The layer's average load.

  For more information about how load is computed, see Load (computing) on Wikipedia.

Crossing a threshold causes a scaling event, upscaling if more instances are needed, and downscaling if fewer instances are needed. OpsWorks Stacks then adds or deletes instances based on the scaling parameters.

Custom CloudWatch alarms

You can use up to five custom CloudWatch alarms as upscaling or downscaling thresholds. They must be in the same region as the stack. For more information about how to create custom alarms, see Creating Amazon CloudWatch Alarms.

> **Note**
>
> To use custom alarms, you must update your service role to allow `cloudwatch:DescribeAlarms`. You can either have OpsWorks Stacks update the role for you the first time you use this feature, or you can edit the role manually. For more information, see Allowing OpsWorks Stacks to Act on Your Behalf.
> When there are multiple alarms configured for load-based configuration, if an alarm is in the `INSUFFICIENT_DATA` metric alarm state, load-based instance scaling cannot occur even if another alarm is in the `ALARM` state. Auto scaling can proceed only if all alarms are in the `OK` or `ALARM` states. For more information about using Amazon CloudWatch alarms, see Using Amazon CloudWatch alarms in the *Amazon CloudWatch User Guide*.

Scaling parameters

The following parameters control how OpsWorks Stacks manages scaling events.

- **Start servers in batches of** – The number of instances to add or remove when the scaling event occurs.
- **If thresholds are exceeded** – The amount of time (in minutes), that the load must remain over an upscaling threshold or under a downscaling threshold before OpsWorks Stacks triggers a scaling event.
- **After scaling, ignore metrics** – The amount of time (in minutes) after a scaling event occurs that OpsWorks Stacks should ignore metrics and suppress additional scaling events.

  For example, OpsWorks Stacks adds new instances following an upscaling event but the instances won't start reducing the load until they have been booted and configured. There is no point in raising additional scaling events until the new instances are online and handling requests, which typically takes several minutes. This setting allows you to direct OpsWorks Stacks to suppress scaling events long enough to get the new instances online.

  You can increase this setting to prevent sudden swings in scaling when layer averages such as **Average CPU**, **Average memory**, or **Average load** are in temporary disagreement.

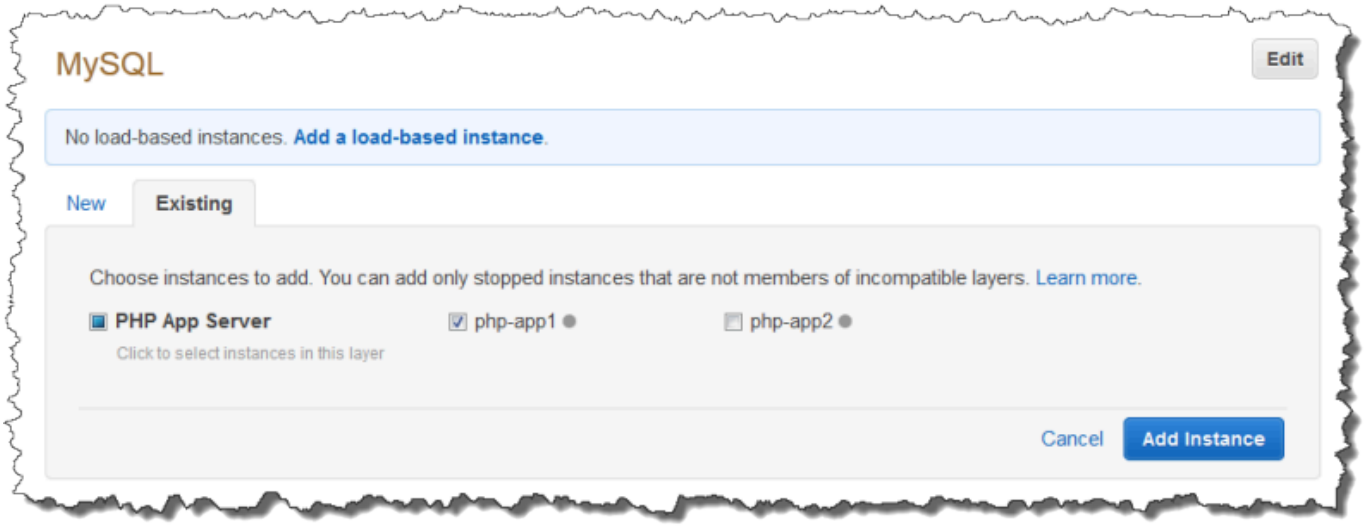
For example, if CPU usage is above the limit and memory usage is close to downscaling, an instance upscale event might immediately be followed by a memory downscaling event. To prevent this, you can increase the number of minutes in the **After scaling, ignore metrics** setting. In this example, the CPU scaling would occur, but the memory downscaling event would not.

3. To add additional load-based instances, choose **+ Instance**, configure the settings, and then choose **Add Instance**. Repeat until you have enough load-based instances to handle your maximum anticipated load. Then choose **Save**.

> **Note**
>
> You can also add a new load-based instance to a layer by opening the **Load-based** page, and choosing **Add a load-based instance** (if you have not yet added a load-based instance to the layer) or **+ Instance** (if the layer already has one or more load-based instances). Then configure the instance as described earlier in this section.

To add an existing load-based instance to a layer

1. In the navigation pane, under **Instances**, choose **Load-based**.
2. If you have already enabled load-based automatic scaling for a layer, choose **+ Instance**. Otherwise, choose **Add a load-based instance**. Choose the **Existing** tab.



3. On the **Existing** tab, choose an instance. The list shows only load-based instances.

> **Note**
>
> If you change your mind about using an existing instance, on the **New** tab, create a new instance as described in the preceding procedure.

4. Choose **Add Instance** to add the instance to the layer.

You can modify the configuration for or disable automatic load-based scaling at any time.

To disable automatic load-based scaling

1. In the navigation pane, under **Instances**, choose **Load-based**, and then choose **edit** for the appropriate layer.
2. Switch **Load-based auto scaling enabled** to **No**.

How load-based scaling differs from auto healing

Automatic load-based scaling uses load metrics that are averaged across all running instances. If the metrics remain between the specified thresholds, OpsWorks Stacks does not start or stop any instances. With auto healing, on the other hand, OpsWorks Stacks automatically starts a new instance with the same configuration when an instance stops responding. The instance may not be able to respond due to a network issue or some problem with the instance.

For example, suppose your CPU upscaling threshold is 80%, and one instance stops responding.

- If auto healing is disabled, and the remaining running instances can keep average CPU utilization below 80%, OpsWorks Stacks does not start a new instance. It starts a replacement instance only if the average CPU utilization across the remaining instances exceeds 80%.
- If auto healing is enabled, OpsWorks Stacks starts a replacement instance regardless of load thresholds.

Using Computing Resources Created Outside of OpsWorks Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate

their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

This feature is supported only for Linux stacks.

Instances describes how to use OpsWorks Stacks to create and manage groups of Amazon Elastic Compute Cloud (Amazon EC2) instances. You can also incorporate Linux computing resources into a stack that was created outside of OpsWorks Stacks:

- Amazon EC2 instances that you created directly by using the Amazon EC2 console, CLI, or API.
- *On-premises* instances running on your own hardware, including instances running in virtual machines.

These computing resources become OpsWorks Stacks-managed instances, and you can manage them much like regular OpsWorks Stacks instances:

- **Manage user permissions** – You can use OpsWorks Stacks user management to specify which users are allowed to access your stacks, which actions they are allowed to perform on the stack's instances, and whether they have SSH access and sudo privileges.
- **Automate tasks** – You can have OpsWorks Stacks run custom Chef recipes to perform tasks such as executing scripts on any or all of a stack's instances with a single command.

  If you assign the instance to a layer, OpsWorks Stacks automatically runs a specified set of Chef recipes on the instance at key points in its lifecycle, including your custom recipes. Note that you can assign registered Amazon EC2 instances to custom layers only.
- **Manage resources** – A stack lets you group and manage resources in an AWS Region, and the OpsWorks dashboard shows the status of your stacks across all Regions.
- **Install packages** – You can use Chef recipes to install packages on any instance in a stack.
- **Update the operating system** – OpsWorks Stacks provides a simple way to install operating system security patches and updates on a stack's instances.

- **Deploy applications** – OpsWorks Stacks deploys applications consistently to all of the stack's application server instances.
- **Monitoring** – OpsWorks Stacks creates custom CloudWatch metrics to monitor all of your stack's instances.

For pricing information, see AWS OpsWorks Pricing.

Following is the basic procedure for working with a registered instance.

1. Register the instance with a stack.

   The instance is now part of the stack and managed by OpsWorks Stacks.
2. Optionally, assign the instance to a layer.

   This step lets you take full advantage of OpsWorks Stacks management functionality. You can assign registered on-premises instances to any layer; registered Amazon EC2 instances can be assigned to custom layers only.
3. Use OpsWorks Stacks to manage the instance.
4. When you no longer need the instance in the stack, deregister it, which removes the instance from OpsWorks Stacks.

The following sections describe this process in detail.

Topics

- Registering an Instance with an OpsWorks Stacks Stack
- Managing Registered Instances
- Assigning a Registered Instance to a Layer
- Unassigning a Registered Instance
- Deregistering a Registered Instance
- Registered Instance Life Cycle

Registering an Instance with an OpsWorks Stacks Stack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This feature is supported only for Linux stacks.

To register an instance that is outside of OpsWorks Stacks, you run the AWS CLI **aws opsworks register** command. You can run this command from the instance that you want to register, or from another computer. You apply the `AWSOpsWorksRegisterCLI_EC2` or `AWSOpsWorksRegisterCLI_OnPremises` policies to a user or group to grant permissions required for the AWS CLI to register EC2 or on-premises instances, respectively. These policies require version 1.16.180 of the AWS CLI or newer.

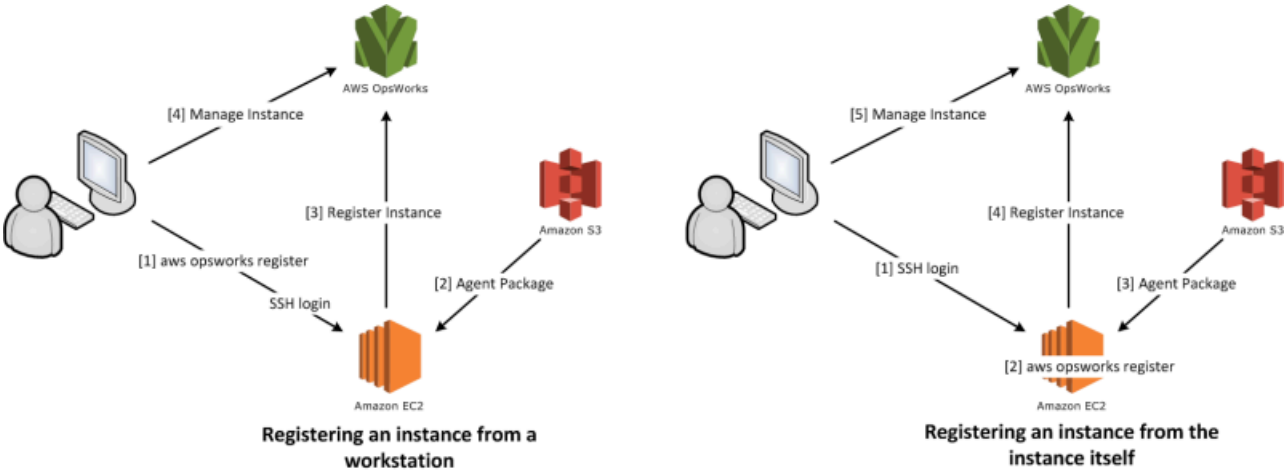
> **Note**
>
> To prevent users or roles from registering instances, update the instance profile to deny access to the **register** command.

The registration process installs an agent on an instance that you want to manage by using OpsWorks Stacks, and registers the instance with an OpsWorks stack that you specify. After you register an instance, the instance is part of the stack and is managed by OpsWorks Stacks. For more information, see Managing Registered Instances.

Note

Although AWS Tools for PowerShell includes the `Register-OpsInstance` cmdlet, which calls the `register` API action, we recommend that you use the AWS CLI to run the `register` command instead.

The following diagram shows both approaches to registering an Amazon EC2 instance. You can use the same approaches for registering an on-premises instance.



Note

You can use the OpsWorks Stacks console to manage a registered instance, but you must run an AWS CLI `register` command to register the instance. The reason for this is that the registration process must be run from the instance, which can't be done by the console.

The following sections describe the procedure in detail.

Topics

- Walkthrough: Register an Instance from Your Workstation
- Registering Amazon EC2 and On-premises Instances

Walkthrough: Register an Instance from Your Workstation

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This feature is supported only for Linux stacks.

The registration process supports several scenarios. This section walks you through an end-to-end example of one scenario: how to use your workstation to register an Amazon EC2 instance. Other registration scenarios use a similar procedure. For more information, see Registering Amazon EC2 and On-premises Instances.

> **Note**
>
> You typically want to register an existing Amazon EC2 instance. However, you can just create a new instance and a new stack for the walkthrough and delete them when you are finished.

Topics

- Step 1: Create a Stack and an Instance
- Step 2: Install and Configure the AWS CLI
- Step 3: Register the Instance with the EC2Register Stack

Step 1: Create a Stack and an Instance

To get started, you need a stack and an Amazon EC2 instance to be registered with that stack.

To create the stack and instance

1. Use the OpsWorks Stacks console to create a new stack named **`EC2Register`**. You can accept default values for the other stack settings.
2. Launch a new instance from the Amazon EC2 console. Note the following.

   - The instance must in the same region and VPC as the stack.

     If you are using a VPC, pick a public subnet for this walkthrough.
   - If you need to create an SSH key, save the private key file to your workstation and record the name and file location.

     If you use an existing key, record the name and private key file location. You need those values later.
   - The instance must be based on one of the supported Linux operating systems. For example, if your stack is in US West (Oregon), you can use `ami-35501205` to launch a Ubuntu 14.04 LTS instance in that region.

   Otherwise, accept the default values.

While the instance is booting, you can proceed to the next section.

Step 2: Install and Configure the AWS CLI

Registration is performed by using the AWS CLI **aws opsworks register** command. Before you register your first instance, you must be running version 1.16.180 of the AWS CLI or newer. The installation details depend on your workstation's operating system. For more information about installing the AWS CLI, see Installing the AWS Command Line Interface. To check the version of the AWS CLI that you are running, enter `aws --version` in a shell session.

> **Note**
>
> To prevent users or roles from registering instances, update the instance profile to deny access to the **register** command.

We strongly recommend that you do not skip this step, even if you are already running the AWS CLI on your workstation. Using the most current release of the AWS CLI is a security best practice.

You must provide `register` with a set of AWS credentials that have appropriate permissions. The recommended way to do this—so that you avoid installing credentials directly on an instance—is to register instances that are launched with an instance profile, and then add the `--use-instance-profile` switch to your `register` command. If you are getting credentials from an instance profile, skip to Step 3: Register the Instance with the EC2Register Stack in this topic. However, if your instance was not launched with an instance profile, you can create an IAM user. The following procedure creates a new user with appropriate permissions, installing the user's credentials on the workstation, and then passing those credentials to `register`.

> **Warning**
>
> IAM users have long-term credentials, which presents a security risk. To help mitigate this risk, we recommend that you provide these users with only the permissions they require to perform the task and that you remove these users when they are no longer needed.

To create the user

1. On the IAM console, choose **Users** in the navigation pane, and then choose **Add user**.
2. Add a user named **`EC2Register`**.
3. Choose **Next**.
4. On the **Set permissions** page, choose **Attach policies directly**.
5. Enter **`OpsWorks`** in the **Permissions policy** filter box to display the OpsWorks policies, select one of the following policies, and then choose **Next: review**. The policy grants your user the permissions that are required to run `register`.
   - Choose `AWSOpsWorksRegisterCLI_EC2` to allow the user permissions to register EC2 instances that use instance profiles.
   - Choose `AWSOpsWorksRegisterCLI_OnPremises` to allow the user permissions to register on-premises instances.
6. Choose **Next**.
7. On the **Review** page, choose **Create user**.
8. Now create access keys for your user. From the navigation pane, choose **Users**, and then choose the user you want to create access keys for.
9. Choose the **Security credentials** tab, then choose **Create access key**.
10. Choose the **Access key best practices & alternatives** that best corresponds to your task.

11. Choose **Next**.
12. (Optionally) enter a tag to identify the access keys.
13. Choose **Next**.
14. Choose **Download .csv file**, save the credentials file to a convenient location on your system, and choose **Done**.

You need to provide the IAM user's credentials to `register`. This walkthrough handles the task by installing the EC2Register credentials in your workstation's `credentials` file. For information about other ways to manage credentials for the AWS CLI, see Configuration and Credential Files.

To install the user's credentials

1. Create or open your workstation's `credentials` file. The file is located at `~/.aws/credentials` (Linux, Unix, and OS X) or `C:\Users\`*`User_Name`*`\.aws\credentials` (Windows systems).
2. Add a profile for the EC2Register user to the `credentials` file, using the following format.

   ```
   [ec2register]
   aws_access_key_id = access_key_id
   aws_secret_access_key = secret_access_key
   ```

   Replace *`access_key_id`* and *`secret_access_key`* with the EC2Register keys for that you downloaded earlier.

Step 3: Register the Instance with the EC2Register Stack

You are now ready to register the instance.

To register the instance

1. In OpsWorks Stacks, return to the EC2Register stack, choose **Instances** in the navigation pane, and then choose **Register an instance**.
2. Select **EC2 Instances**, choose **Next: Select Instances**, and select your instance from the list.
3. Choose **Next: Install AWS CLI**, and **Next: Register Instances**. OpsWorks Stacks automatically uses the available information, such as the stack ID and the instance ID to create a `register` command template, which is displayed on the **Register Instances** page. For this example, you

use `register` to log in to the instance with an SSH key and explicitly specify the key file, so set **I use SSH keys to connect to my instances** to **Yes**. The command template resembles the following.

```
aws opsworks register --infrastructure-class ec2 --region region endpoint ID
  --stack-id 247be7ea-3551-4177-9524-1ff804f453e3 --ssh-username [username]
  --ssh-private-key [key-file] i-f1245d10
```

> **Note**
>
> You must set the Region to the OpsWorks Stacks service's endpoint Region, not the stack's Region, if the stack is within a classic Region associated with the `us-east-1` regional endpoint. OpsWorks Stacks determines the stack's Region from the stack ID.

4. The command template contains several user-specific argument values, which are indicated by brackets and must be replaced with appropriate values. Copy the command template to a text editor and edit it as follows.

> **Important**
>
> The IAM user that is created during the registration process is required throughout the life of a registered instance. Deleting the user causes the OpsWorks Stacks agent to be unable to communicate with the service. To help prevent problems managing registered instances in the event that the user is accidentally deleted, add the `--use-instance-profile` parameter to your `register` command to use the instance's built-in instance profile instead. Adding the `--use-instance-profile` parameter also prevents errors from occurring when you rotate AWS account access keys every 90 days (a recommended best practice), because it prevents mismatches between the access keys available to the OpsWorks agent and required IAM user.

- Replace *`key file`* with the fully qualified path of the private key file for the Amazon EC2 key pair that you saved when you created the instance.

  You can use a relative path, if you prefer.

- Replace *`username`* with the instance's user name.

For this example, the user name is either `ubuntu`, for an Ubuntu instance, or `ec2-user`, for a Red Hat Enterprise Linux (RHEL) or Amazon Linux instance.

- Add `--use-instance-profile`, which runs `register` with the instance profile to prevent errors during key rotation or if the principal IAM user is accidentally deleted.

Your command should resemble the following.

```
aws opsworks register --use-instance-profile --infrastructure-class ec2 \
  --region us-west-2  --stack-id 247be7ea-3551-4177-9524-1ff804f453e3 --ssh-
username ubuntu \
  --ssh-private-key "./keys/mykeys.pem" i-f1245d10
```

5. Open a terminal window on your workstation, paste the `register` command from your editor, and run the command.

   Registration typically takes around five minutes. When it is complete, return to the OpsWorks Stacks console and choose **Done**. Then choose **Instances** in the navigation pane. Your instance should be listed under **Unassigned Instances**. You can then assign the instance to a layer or leave it where it is, depending on how you intend to manage the instance.

6. When you are finished, stop the instance and then delete it by using the OpsWorks Stacks console or commands. This terminates the Amazon EC2 instance, so you don't incur any further charges.

Registering Amazon EC2 and On-premises Instances

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This feature is supported only for Linux stacks.

This section describes how to register an Amazon EC2 or on-premises instance with an OpsWorks Stacks stack.

Topics

- Preparing the Instance
- Installing and Configuring the AWS CLI
- Registering the Instance
- Using the register Command
- Example register Commands
- Instance Registration Policies

Preparing the Instance

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This feature is supported only for Linux stacks.

Before registering an instance, you must ensure that it is compatible with OpsWorks Stacks. The details depend on whether you are registering an on-premises or Amazon EC2 instance.

On-premises Instances

An on-premises instance must satisfy the following criteria:

- The instance must run one of the supported Linux operating systems. While it might be possible to create or register instances with other operating systems (such as CentOS 6.*x*) that have been created from custom or community-generated AMIs, these are not officially supported.

  You must install the `libyaml` package on the instance. For Ubuntu instances, the package is named `libyaml-0-2`. For CentOS and Red Hat Enterprise Linux instances, the package is named `libyaml`.
- The instance must have a supported instance type (sometimes called the instance size). Supported instance types can vary by operating system, and depend on whether your stack is in a VPC. For a list of supported instance types, view the **Size** drop-down list values that are shown in the OpsWorks Stacks console when you try to create a new instance in your target stack. If an instance type is grayed-out, and cannot be created in your target stack, then you cannot register an instance of that type.
- The instance must have Internet access that allows it to communicate with the OpsWorks Stacks service endpoint, `opsworks.us-east-1.amazonaws.com (HTTPS)`. The instance must also support outbound connections to AWS resources such as Amazon S3.
- If you plan to register the instance from a separate workstation, the registered instance must support SSH login from the workstation.

  SSH login is not required if you run the registration command from the instance.
- The AWS access key is used for authentication from the OpsWorks agent to the OpsWorks Stacks service. If you rotate access keys as recommended every 90 days, update the OpsWorks agent manually to use the new key. On an on-premises computer or instance, edit the `/etc/aws/opsworks/instance-agent.yml` file with the new access key and secret key. The following command shows the access key and secret key in this file. An agent that is using old keys can cause errors.

  ```
  cat /etc/aws/opsworks/instance-agent.yml | egrep "access_key|secret_key"
  :access_key_id: AKIAIOSFODNN7EXAMPLE
  :secret_access_key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
  ```

Amazon EC2 Instances

An Amazon EC2 instance must satisfy the following criteria:

- The AMI must be based on one of the supported Linux operating systems. For a current list, see OpsWorks Stacks operating systems.

  For more information, see Using Custom AMIs.

  If the instance is based on a custom AMI that derives from a standard supported AMI, or if the instance contains a very minimal setup, you must install the `libyaml` package on the instance. For Ubuntu instances, the package is named `libyaml-0-2`. For Amazon Linux and Red Hat Enterprise Linux instances, the package is named `libyaml`.
- The instance must have a supported instance type (sometimes called the instance size). Supported instance types can vary by operating system, and depend on whether your stack is in a VPC. For a list of supported instance types, view the **Size** drop-down list values that are shown in the OpsWorks Stacks console when you try to create a new instance in your target stack. If an instance type is grayed-out, and cannot be created in your target stack, then you cannot register an instance of that type, either.
- The instance must be in the `running` state.
- The instance should not be part of an Auto Scaling group.

  For more information, see Detach EC2 Instances From Your Auto Scaling Group.
- The instance can be part of a VPC, but it must be in the same VPC as the stack and the VPC must be configured to work properly with OpsWorks Stacks.
- Spot instances are not supported, because they do not work with auto healing.

When you register an Amazon EC2 instance, OpsWorks Stacks does not modify the instance's security groups or rules. Be sure that the instance's security group rules match the following OpsWorks Stacks requirements.

Ingress Rules

Ingress rules should allow the following.

- SSH login.
- Traffic from the appropriate layers.

For example, a database server would typically allow inbound traffic from the stack's application server layers.

- Traffic to the appropriate ports.

  For example, application server instances typically allow all inbound traffic to ports 80 (HTTP) and 443 (HTTPS).

Egress Rules

Egress rules should allow the following.

- Traffic to the OpsWorks Stacks service from applications running on the instance.
- Traffic to access AWS resources such as Amazon S3 from applications using the AWS API.

One common approach is to not specify any egress rules, which places no restrictions on outbound traffic.

Installing and Configuring the AWS CLI

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Before you register your first instance, you must be running version 1.16.180 of the AWS CLI or newer on the computer from which you run `register`. The installation details depend on your workstation's operating system. For more information about installing the AWS CLI, see Installing the AWS Command Line Interface and Configuring the AWS Command Line Interface. To check the version of the AWS CLI that you are running, enter `aws --version` in a shell session.

> **Note**
>
> Although AWS Tools for PowerShell includes the `Register-OpsInstance` cmdlet, which calls the `register` API action, we recommend that you use the AWS CLI to run the `register` command instead.

You must run `register` with appropriate permissions. You can get permissions by using an IAM role, or less optimally, by installing user credentials with appropriate permissions on the workstation or instance to be registered. You can then run `register` with those credentials, as described later. Specify permissions by attaching an IAM policy to the user or role. For `register`, you use either the `AWSOpsWorksRegisterCLI_EC2` or `AWSOpsWorksRegisterCLI_OnPremises` policies, which grant permissions to register Amazon EC2 or on-premises instances, respectively.

> **Note**
>
> If you run `register` on an Amazon EC2 instance, you should ideally use an IAM role to provide credentials. For more information about how to attach an IAM role to an existing instance, see Attach an IAM role to an instance or Replace an IAM role in the *Amazon EC2 User Guide*.

For example snippets of the `AWSOpsWorksRegisterCLI_EC2` and `AWSOpsWorksRegisterCLI_OnPremises` policies, see Instance Registration Policies. For more information about creating and managing AWS credentials, see AWS Security Credentials.

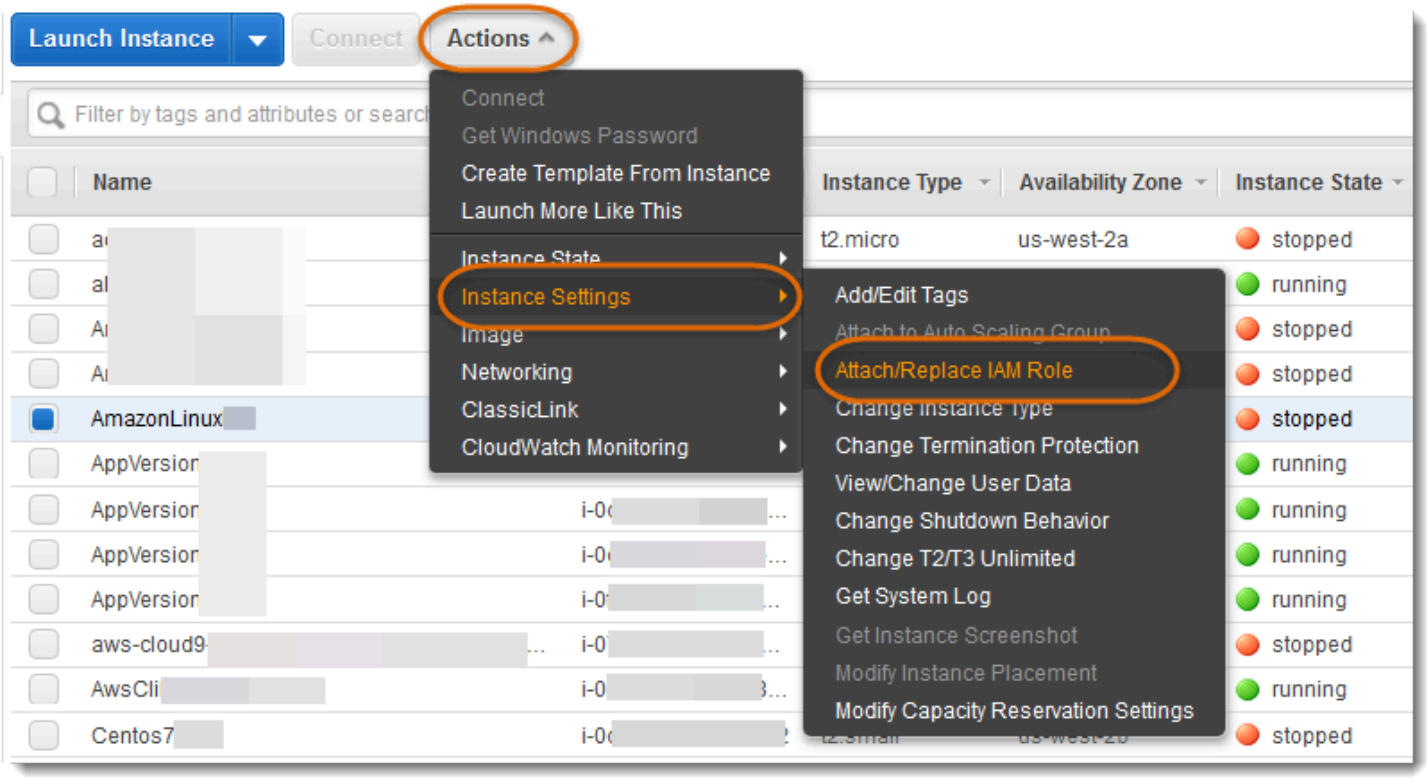
Topics

- Using an IAM Role
- Using Installed Credentials

Using an IAM Role

If you are running the command from the Amazon EC2 instance that you intend to register, the preferred strategy for providing credentials to `register` is to use an IAM role that has the `AWSOpsWorksRegisterCLI_EC2` policy or equivalent permissions attached. This approach allows

you to avoid installing your credentials on the instance. One way to do this is by using the **Attach/Replace IAM Role** command in the EC2 console, as shown in the following image.



For more information about how to attach an IAM role to an existing instance, see Attach an IAM role to an instance or Replace an IAM role in the *Amazon EC2 User Guide*. For instances that were launched with an instance profile (recommended), add the `--use-instance-profile` switch to your `register` command to provide credentials; do not use the `--profile` parameter.

If the instance is running and has a role, you can grant the required permissions by attaching the `AWSOpsWorksRegisterCLI_EC2` policy to the role. The role provides a set of default credentials for the instance. As long as you have not installed any credentials on the instance, `register` automatically assumes the role and runs with its permissions.

> **Important**
>
> We recommend that you do not install credentials on the instance. In addition to creating a security risk, the instance's role is at the end of the default providers chain that the AWS CLI uses to locate the default credentials. Installed credentials might take precedence over the role, and `register` might therefore not have the required permissions. For more information, see Getting started with the AWS CLI.

If a running instance does not have a role, you must install credentials with the required permissions on the instance, as described in Using Installed Credentials. It is recommended, easier, and less error-prone to use instances that are launched with an instance profile.

Using Installed Credentials

There are several ways to install user credentials on a system and provide them to an AWS CLI command. The following describes an approach that is no longer recommended, but can be used if you are registering EC2 instances that were launched without an instance profile. You can also use an existing user's credentials as long as the attached policies grant the required permissions. For more information, including a description of other ways to install credentials, see Configuration and Credential Files.

To use installed credentials

1. Create an IAM user and save the access key ID and secret access key in a secure location.

   > **Warning**
   >
   > IAM users have long-term credentials, which presents a security risk. To help mitigate this risk, we recommend that you provide these users with only the permissions they require to perform the task and that you remove these users when they are no longer needed.

2. Attach the AWSOpsWorksRegisterCLI_OnPremises policy to the user. If you prefer, you can attach a policy that grants broader permissions, as long as it includes the `AWSOpsWorksRegisterCLI_OnPremises` permissions.
3. Create a profile for the user in the system's `credentials` file. The file is located at `~/.aws/credentials` (Linux, Unix, and OS X) or `C:\Users\`*`User_Name`*`\.aws\credentials` (Windows systems). The file contains one or more profiles in the following format, each of which contains a user's access key ID and secret access key.

   ```
   [profile_name]
   aws_access_key_id = access_key_id
   aws_secret_access_key = secret_access_key
   ```

   Substitute the IAM credentials that you saved earlier for the *`access_key_id`* and *`secret_access_key`* values. You can specify any name you prefer for a profile name, with

two limitations: the name must be unique, and the default profile must be named `default`. You can also use an existing profile, as long as it has the required permissions.

4. Use the `register` command's `--profile` parameter to specify the profile name. The `register` command runs with the permissions that are granted to the associated credentials.

   You can also omit `--profile`. In that case, `register` runs with default credentials. Be aware that these are not necessarily the default profile's credentials , so you must ensure that the default credentials have the required permissions. For more information about how the AWS CLI determines default credentials, see Configuring the AWS Command Line Interface.

Registering the Instance

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This feature is supported only for Linux stacks.

You register an instance by running the AWS CLI `register` command from your workstation or from the instance. The simplest way to handle the operation is to use the OpsWorks Stacks console's registration wizard, which simplifies the process of constructing the command string. After you are familiar with the registration procedure, you can skip the wizard if you prefer, and run the `register` command.

The following describes how to use the registration wizard to register an instance with an existing stack.

> **Note**
>
> To register an instance with a new stack, you can do so by choosing **Register Instances** on the OpsWorks Stacks dashboard. This starts a wizard that is identical to the one for existing stacks, except for an additional page that configures the new stack.

To use the registration wizard to register an instance

1. In the OpsWorks Stacks console, create a stack or open an existing stack.
2. Choose **Instances** in the navigation pane, and then choose **register an instance**.
3. On the **Choose an Instance Type** page, specify whether you want to register an Amazon EC2 or an on-premises instance:
   - If you are registering an Amazon EC2 instance, choose **Next: Select Instances**.
   - If you are registering an on-premises instance, choose **Next: Install AWS CLI**, and then go to Step 5.
4. If you are registering an Amazon EC2 instance, open the **Select Instances** page to select the instance to register. OpsWorks Stacks collects the information needed to build the command. When you are finished, choose **Next:Install AWS CLI**.
5. The instance on which you plan to run `register` must be running version 1.16.180 of the AWS CLI or newer. To install or update the AWS CLI, the registration wizard page provides links to installation and configuration instructions. After you have verified the AWS CLI installation, choose whether you are running the command from the instance to be registered or from a separate workstation, and then choose **Next: Register Instances**.
6. The **Register Instances** page displays a template for a `register` command string that incorporates your selected options. For example, if you are registering an Amazon EC2 instance from a separate workstation, the default template resembles the following.

```
aws opsworks register --infrastructure-class ec2 --region us-west-2
  --stack-id 247be7ea-3551-4177-9524-1ff804f453e3 --ssh-username [username] i-
f1245d10
```

> **Important**
>
> The IAM user that is created during the registration process is required throughout the life of a registered instance. Deleting the user causes the OpsWorks Stacks agent to be unable to communicate with the service. To help prevent problems managing registered instances in the event that the user is accidentally deleted, add the `--use-instance-profile` parameter to your `register` command to use the instance's built-in instance profile instead. Adding the `--use-instance-profile` parameter also prevents errors from occurring when you rotate AWS account access keys every 90 days (a recommended best practice), because it prevents mismatches between the access keys available to the OpsWorks agent and required IAM user.

If you set **I use SSH keys** to **Yes**, OpsWorks Stacks adds the `--ssh-private-key` argument to the string, which you can use to specify a private SSH key file.

> **Note**
>
> If you want `register` to log on with a password, set **I use SSH keys** to **No**. When you run `register`, you are prompted for the password.

Copy this string to a text editor. and edit it as required. Note the following.

- The bracketed text represents information that you must supply, such as the location of your SSH key file.
- The template assumes that you are running `register` with default AWS credentials. If not, add a `--profile` argument to the command string, and specify the credential profile name that you want to use.

For other scenarios, you might need to change the command further. For an explanation of the available `register` arguments and alternative ways to construct the command string, see Using the `register` Command. You can also display the command's documentation by running `aws opsworks help register` from the command line. For some example command strings, see Example register Commands.

7. After you have finished editing the command string, open a terminal window on your workstation or use SSH to log on to the instance and run the command. The entire operation typically takes around five minutes, during which the instance is in the **Registering** state.
8. When the operation is finished, choose **Done**. The instance is now in the **Registered** state and is listed as an unassigned instance on the stack's **Instances** page.

The `register` command does the following.

1. If `register` is running on a workstation, the command first uses SSH to log in to the instance to be registered.

   The remainder of the process takes place on the instance, and is the same regardless of where you ran the command.
2. Downloads the OpsWorks Stacks agent package from Amazon S3.
3. Unpacks and installs the agent and its dependencies, such as the AWS SDK for Ruby.
4. Creates the following:
   - An IAM user that bootstraps the agent with the OpsWorks Stacks service to provide secure communication.

     The user's permissions allow only the `opsworks:RegisterInstance` action, and they expire after 15 minutes.
   - An IAM group for the stack, which contains the registered instances' users.
5. Creates an RSA key pair and sends the public key to OpsWorks Stacks.

   This key pair is used to encrypt communications between the agent and OpsWorks Stacks.
6. Registers the instance with OpsWorks Stacks. The stack then runs a set of initial setup recipes to configure the instance, which includes the following.
   - Overwriting the instance's hosts file.

     By registering the instance, you have handed user management over to OpsWorks Stacks, which must have its own hosts file to control SSH login permissions.
   - For Amazon EC2 instances, initial setup also includes registering any attached Amazon EBS volumes or Elastic IP addresses with the stack.

     You must ensure that the Amazon EBS volumes are not mounted to reserved mount points, including `/var/www` and any mount points that are reserved by the instance's layers. For

more information about managing stack resources, see Resource Management. For more information about layer mount points, see OpsWorks Stacks Layer Reference.

For a complete description of the initial setup configuration changes, see Initial Setup Configuration Changes.

> **Note**
>
> Initial setup does not update a registered instance's operating system; you must handle that task yourself. For more information, see Managing Security Updates.

Using the `register` Command

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This feature is supported only for Linux stacks.

To register an instance, be sure you are running at least version 1.16.180 of the AWS CLI. The following shows the general syntax for the `register` command.

```
aws opsworks register \
  [--profile profile_name] \
  [--region region_name] \
  --infrastructure-class instance_type \
  --stack-id stack ID \
  [--local] | [--ssh-private-key key_file --ssh-username username] | [--override-
ssh command_string] \
```

```
  [--override-hostname hostname] \
  [--debug] \
  [--override-public-ip public IP] \
  [--override-private-ip private IP] \
..[--use-instance-profile] \
  [ [IP address] | [hostname] | [instance ID]
```

The following arguments are common to all AWS CLI commands.

`--profile`

(Optional) The credential's profile name. If you omit this argument, the command runs with your default credentials. For more information about how the AWS CLI determines default credentials, see Configuring the AWS Command Line Interface.

`--region`

(Optional) The OpsWorks Stacks service endpoint's Region. Do not set `--region` to the stack's Region. OpsWorks Stacks automatically determines the stack's Region from the stack ID.

> **Note**
>
> If your default Region is already set, you can omit this argument. For more information about how to specify a default Region, see Configuring the AWS Command Line Interface.

Use the following arguments for both Amazon EC2 and on-premises instances.

`--infrastructure-class`

(Required) This parameter must be set to either `ec2` or `on-premises`, to indicate whether you are registering an Amazon EC2 or on-premises instance, respectively.

`--stack-id`

(Required) The ID of the stack that the instance is to be registered with.

> **Note**
>
> To find a stack ID, on the **Stack** page, choose **Settings**. The stack ID is labeled **OpsWorks ID**, and is a GUID that looks something like `ad21bce6-7623-47f1-bf9d-af2affad8907`.

SSH Login Arguments

Use the following arguments to specify how `register` should log in to the instance.

`--local`

(Optional) Use this argument to register the instance that you run the command on.

In this case, `register` does not need to log in to the instance.

`--ssh-private-key and --ssh-username`

(Optional) Use these arguments if you are registering the instance from a separate workstation and want to explicitly specify the user name or private key file.

- `--ssh-username` – Use this argument to specify an SSH user name.

  If you omit `--ssh-username`, `ssh` uses the default user name.
- `--ssh-private-key` – Use this argument to explicitly specify a private key file.

  If you omit `--ssh-private-key`, `ssh` attempts to log in using authentication techniques that do not require a password, including using the default private key. If none of those techniques are supported, `ssh` queries for your password. For more information about how `ssh` handles authentication, see The Secure Shell (SSH) Authentication Protocol.

`--override-ssh`

(Optional) Use this argument if you are registering the instance from a separate workstation and want to specify a custom `ssh` command string. The `register` command uses this command string to log in to the registered instance.

For more information about `ssh`, see SSH.

`--override-hostname`

(Optional) Specifies a host name for the instance, which is used only by OpsWorks Stacks. The default value is the instance's host name.

`--debug`

(Optional) Provides debugging information if the registration process fails. For troubleshooting information, see Troubleshooting Instance Registration.

`--use-instance-profile`

(Optional, but highly recommended for Amazon EC2 instances) Lets the `register` command use an attached instance profile, instead of creating an IAM user. Adding this parameter can help prevent errors that occur if you try to manage a registered instance when the IAM user has accidentally been deleted.

> **Important**
>
> The IAM user that is created during the registration process is required throughout the life of a registered instance. Deleting the user causes the OpsWorks Stacks agent to be unable to communicate with the service. To help prevent problems managing registered instances in the event that the user is accidentally deleted, add the `--use-instance-profile` parameter to your `register` command to use the instance's built-in instance profile instead. Adding the `--use-instance-profile` parameter also prevents errors from occurring when you rotate AWS account access keys every 90 days (a recommended best practice), because it prevents mismatches between the access keys available to the OpsWorks agent and required user.

Target

(Conditional) If you run this command from a workstation, the final value in the command string specifies the registration target in one of the following ways.

- The instance's public IP address.
- The instance's host name.
- For Amazon EC2 instances, the instance ID.

  OpsWorks Stacks uses the instance ID to obtain the instance configuration, including the instance's public IP address. By default, OpsWorks Stacks uses this address to construct the `ssh` command string that it uses to log in to the instance. If you need to connect to a private IP address, you must use `--override-ssh` to provide a custom command string. For an example, see Register an On-Premises Instance from a Workstation.

> **Note**
>
> If you specify a host name, `ssh` depends on the DNS server to resolve the name to a particular instance. If you aren't certain that the host name is unique, use `ssh` to verify that the host name resolves to the correct instance.

If you run this command from the instance to be registered, omit the instance identifier and instead use the `--local` argument.

The following arguments are only for on-premises instances.

`--override-public-ip`

(Optional) OpsWorks Stacks displays the specified address as the instance's public IP address. It does not change the instance's public IP address. However, if a user uses the console to connect to the instance, such as by choosing the address on the **Instances** page, OpsWorks Stacks uses the specified address. OpsWorks Stacks automatically determines the argument's default value.

`--override-private-ip`

(Optional) OpsWorks Stacks displays the specified address as the instance's private IP address. It does not change the instance's private IP address. OpsWorks Stacks automatically determines the argument's default value.

Example register Commands

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This feature is supported only for Linux stacks.

This section contains some examples of `register` command strings.

Register an Amazon EC2 Instance from a Workstation

The following example registers an Amazon EC2 instance from a workstation. The command string uses default credentials and identifies the instance by its Amazon EC2 instance ID. You can use the example for on-premises instances by changing `ec2` to `on-premises`.

```
aws opsworks register \
  --region us-west-2 \
  --use-instance-profile \
  --infrastructure-class ec2 \
  --stack-id ad21bce6-7623-47f1-bf9d-af2affad8907 \
  --ssh-user-name my-sshusername \
  --ssh-private-key "./keys/mykeys.pem" \
  i-2422b9c5
```

Register an On-Premises Instance from a Workstation

The following example registers an on-premises instance from a separate workstation. The command string uses default credentials and logs in to the instance with the specified `ssh` command string. If your instance requires a password, `register` prompts you. You can use the example for Amazon EC2 instances by changing `on-premises` to `ec2`.

```
aws opsworks register \
  --region us-west-2 \
  --infrastructure-class on-premises \
  --stack-id ad21bce6-7623-47f1-bf9d-af2affad8907 \
  --override-ssh "ssh your-user@192.0.2.0"
```

> **Note**
>
> You can use `--override-ssh` to specify any custom SSH command string. OpsWorks Stacks then uses the specified string to log in to the instance instead of constructing

a command string. For another example, see Register an Instance Using a Custom SSH Command String.

Register an Instance Using a Custom SSH Command String

The following example registers an on-premises instance from a workstation, and uses the `--override-ssh` argument to specify a custom SSH command that `register` uses to log in to the instance. This example uses `sshpass` to log in with a user name and password, but you can specify any valid `ssh` command string.

```
aws opsworks register \
  --region us-west-2 \
  --infrastructure-class on-premises \
  --stack-id 2f92ff9d-04f2-4728-879b-f4283b40783c \
  --override-ssh "sshpass -p 'mypassword' ssh your-user@192.0.2.0"
```

Register an Instance by Running `register` from the Instance

The following example shows how to register an Amazon EC2 instance by running `register` from the instance itself. The command string depends on default credentials for its permissions. To use the example for an on-premises instance, change `--infrastructure-class` to `on-premises`.

```
aws opsworks register \
  --region us-west-2 \
  --infrastructure-class ec2 \
  --stack-id ad21bce6-7623-47f1-bf9d-af2affad8907 \
  --local
```

Register an Instance with a Private IP Address

By default, `register` uses the instance's public IP address to log in to the instance. To register an instance with a private IP address, such as an instance in a VPC's private subnet, you must use `--override-ssh` to specify a custom `ssh` command string.

```
aws opsworks register \
  --region us-west-2 \
  --infrastructure-class ec2 \
  --stack-id 2f92ff9d-04f2-4728-879b-f4283b40783c \
```

```
--override-ssh "ssh -i mykey.pem ec2-user@10.183.201.93" \
i-2422b9c5
```

Instance Registration Policies

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The `AWSOpsWorksRegisterCLI_EC2` and `AWSOpsWorksRegisterCLI_OnPremises` policies provide the correct permissions for registering EC2 and on-premises instances, respectively. You add `AWSOpsWorksRegisterCLI_EC2` to your IAM user to register EC2 instances, but add `AWSOpsWorksRegisterCLI_OnPremises` to your user to register on-premises instances. To use these policies, you must be running at least version 1.16.180 of the AWS CLI or newer.

The `AWSOpsWorksRegisterCLI_EC2` Policy

Add `AWSOpsWorksRegisterCLI_EC2` to your user to register EC2 instances. You should use this profile if you plan to register only EC2 instances. When you use this policy, permissions are provided by the EC2 instance's instance profile.

JSON

```
{
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "opsworks:AssignInstance",
            "opsworks:CreateLayer",
            "opsworks:DeregisterInstance",
            "opsworks:DescribeInstances",
            "opsworks:DescribeStackProvisioningParameters",
```

```
            "opsworks:DescribeStacks",
            "opsworks:UnassignInstance"
          ],
          "Resource": [
            "*"
          ]
        },
        {
          "Effect": "Allow",
          "Action": [
            "ec2:DescribeInstances"
          ],
          "Resource": [
            "*"
          ]
        }
      ]
    }
```

(Deprecated) The AWSOpsWorksRegisterCLI_OnPremises Policy

Add `AWSOpsWorksRegisterCLI_OnPremises` to your user to register on-premises instances. This policy includes IAM permissions, such as `AttachUserPolicy`, but the resources on which those permissions work are restricted.

JSON

```
    {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": [
            "opsworks:AssignInstance",
            "opsworks:CreateLayer",
            "opsworks:DeregisterInstance",
            "opsworks:DescribeInstances",
            "opsworks:DescribeStackProvisioningParameters",
            "opsworks:DescribeStacks",
            "opsworks:UnassignInstance"
          ],
```

```
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:CreateGroup",
        "iam:AddUserToGroup"
      ],
      "Resource": [
        "arn:aws:iam::*:group/AWS/OpsWorks/OpsWorks-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:CreateUser",
        "iam:CreateAccessKey"
      ],
      "Resource": [
        "arn:aws:iam::*:user/AWS/OpsWorks/OpsWorks-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:AttachUserPolicy"
      ],
      "Resource": [
        "arn:aws:iam::*:user/AWS/OpsWorks/OpsWorks-*"
      ],
      "Condition": {
        "ArnEquals":
          {
```

```
                "iam:PolicyARN": "arn:aws:iam::aws:policy/
AWSOpsWorksInstanceRegistration"
              }
            }
        }
      ]
    }
```

(Deprecated) The AWSOpsWorksRegisterCLI Policy

> **Important**
>
> The AWSOpsWorksRegisterCLI policy has been deprecated, and cannot be used to register new instances. It is available only for backward compatibility on instances that have already been registered. The AWSOpsWorksRegisterCLI policy includes many IAM permissions including CreateUser, PutUserPolicy, and AddUserToGroup. Because these are administrator-level permissions, you should only assign the AWSOpsWorksRegisterCLI policy to trusted administrative users.

Managing Registered Instances

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This feature is supported only for Linux stacks.

When you register an instance, it becomes an OpsWorks Stacks instance, and you can manage it in much the same way as instances created with OpsWorks Stacks. There are two primary differences:

- Registered instances do not have to be assigned to a layer.
- You can deregister a registered instance and return it to your direct control.

After you register an instance, it is in the Registered state. OpsWorks Stacks provides the following management functionality to all registered instances:

- **Health checks** – OpsWorks Stacks monitors the agent to evaluate whether the instance continues to function.

  If an instance fails a health check, OpsWorks Stacks autoheals registered Amazon EC2 instances and changes the status of registered on-premises instances to `connection lost`.
- **CloudWatch monitoring** – CloudWatch monitoring is enabled for registered instance.

  You can monitor metrics such as CPU utilization and available memory and optionally receive a notification if a metric crosses a specified threshold.
- **User management** – OpsWorks Stacks provides a simple way to specify which users can access the instance and what operations they are allowed to perform. For more information, see Managing User Permissions.
- **Recipe execution** – You can use the Execute Recipes stack command to execute Chef recipes on the instance.
- **Operating system updates** – You can use the Update Dependencies stack command to update the instance's operating system.

To take full advantage of OpsWorks Stacks management functionality, you can assign the instance to a layer. For more information, see Assigning a Registered Instance to a Layer.

There are differences between the way OpsWorks Stacks manages Amazon EC2 and on-premises instances.

Amazon EC2 Instances

- If you stop a registered Amazon EC2 instance, OpsWorks Stacks terminates instance store-backed instances and stops Amazon EBS-backed instances.

  The instance is still registered with the stack and assigned to its layers, so you can restart it if needed. You must deregister a registered instance to remove it from a stack, either explicitly or by deleting the instance, which automatically deregisters it.

- If you restart a registered Amazon EC2 instance or the instance fails and is autohealed, the result is the same as stopping and restarting the instance by using Amazon EC2. Note these differences:
  - Instance store-backed instances – OpsWorks Stacks starts a new instance with the same AMI.

    Note that OpsWorks Stacks has no knowledge of any operations that you performed on the instance before it was registered, such as installing software packages. If you want OpsWorks Stacks to install packages or perform other configuration tasks at startup, you must provide custom Chef recipes that perform the required tasks and assign them to the appropriate layers' Setup events.
  - Amazon EBS-backed instances – OpsWorks Stacks starts a new instance with the same AMI and reattaches the root volume, which restores the instance to its previous configuration.
- If you deregister a registered Amazon EC2 instance, it returns to being a regular Amazon EC2 instance.

On-premises Instances

- OpsWorks Stacks cannot stop or start a registered on-premises instance.

  Unassigning a registered on-premises instance triggers a Shutdown event. However, that event simply runs the assigned layers' Shutdown recipes. They perform tasks such as shutting down services, but do not stop the instance.
- OpsWorks Stacks cannot autoheal a registered on-premises instance if it fails, but the instance is marked as connection lost.
- On-premises instances cannot use the Elastic Load Balancing, Amazon EBS, or Elastic IP address services.

Assigning a Registered Instance to a Layer

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This feature is supported only for Linux stacks.

After you register an instance, you can assign it to one or more layers. The advantage of assigning an instance to a layer instead of leaving it unassigned is that you can assign custom recipes to the layer's lifecycle events. OpsWorks Stacks then runs them automatically at the appropriate time, after the layer's recipes for that event.

- You can assign any registered instance to a custom layer. A custom layer has a minimal set of recipes that do not install any packages, so they should not create any conflicts with the instance's existing configuration.
- You can assign on-premises instances to OpsWorks Stacks built-in layers.

  Every built-in layer includes recipes that automatically install one or more packages. For example, the Java App Server Setup recipes install Apache and Tomcat. The layer's recipes might also perform other operations such as restarting services and deploying applications. Before assigning an on-premises instance to a built-in layer, you should ensure that the layer's recipes do not create any conflicts, such as attempting to install a different application server version than is currently on the instance. For more information, see Layers and OpsWorks Stacks Layer Reference.

To assign a registered instance to a layer

1. Add the layers that you want to use to the stack, if you have not done so already.
2. Choose **Instances** in the navigation pane and then choose **assign** in the instance's **Actions** column.
3. Select the appropriate layers and choose **Save**.

When you assign an instance to a layer OpsWorks Stacks does the following.

- Runs the layer's Setup recipes.
- Adds any attached Elastic IP addresses or Amazon EBS volumes to the stack's resources.

  You can then use OpsWorks Stacks to manage these resources. For more information, see Resource Management.

After they are finished, the instance is in the online status and fully incorporated into the stack. OpsWorks Stacks then runs the layer's assigned recipes each time a lifecycle event occurs.

Unassigning a Registered Instance

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

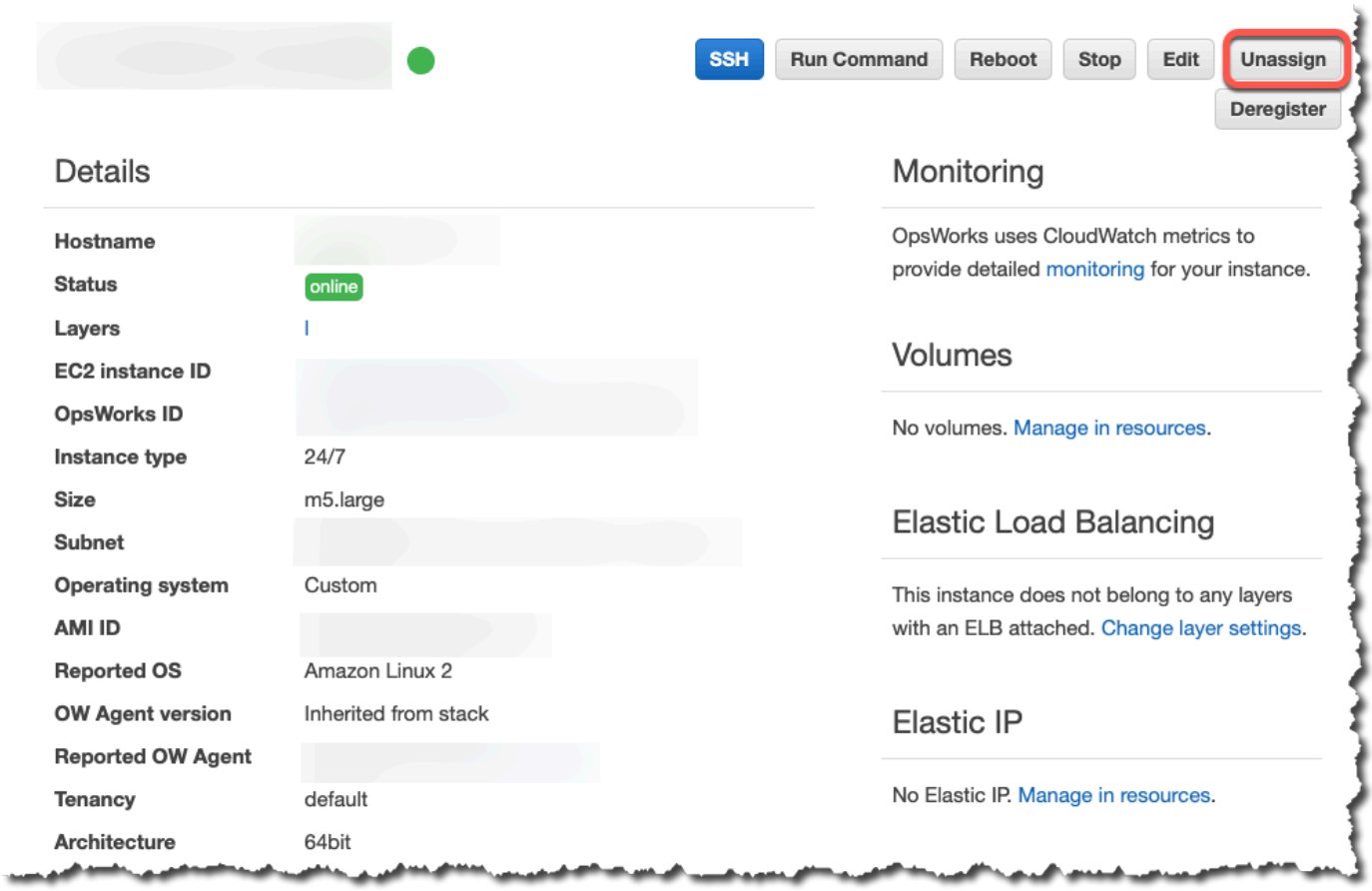
> **Note**
>
> This feature is supported only for Linux stacks.

You can unassign a registered instance from its layers by using the OpsWorks console, AWS CLI, or SDK operation.

When you unassign an instance, OpsWorks Stacks runs the layer's Shutdown recipes on the instance. These recipes perform tasks such as shutting down services but do not stop the instance. If the instance is assigned to multiple layers, unassign applies to every layer; you can't unassign an instance from a subset of its layers. However, the instance is still registered with the stack, and you can assign it to another layer if you wish.

To unassign a registered instance using the console

1. In the navigation pane, choose **Instances**.
2. Choose the instance that you want to unassign.
3. On the **Details** page for the instance, choose **Unassign**.



To unassign a registered instance using the AWS CLI

Run the **aws opsworks unassign-instance** command to unassign a registered instance from all layers that are using the instance.

```
aws opsworks unassign-instance --region region --instance-id instance-id
```

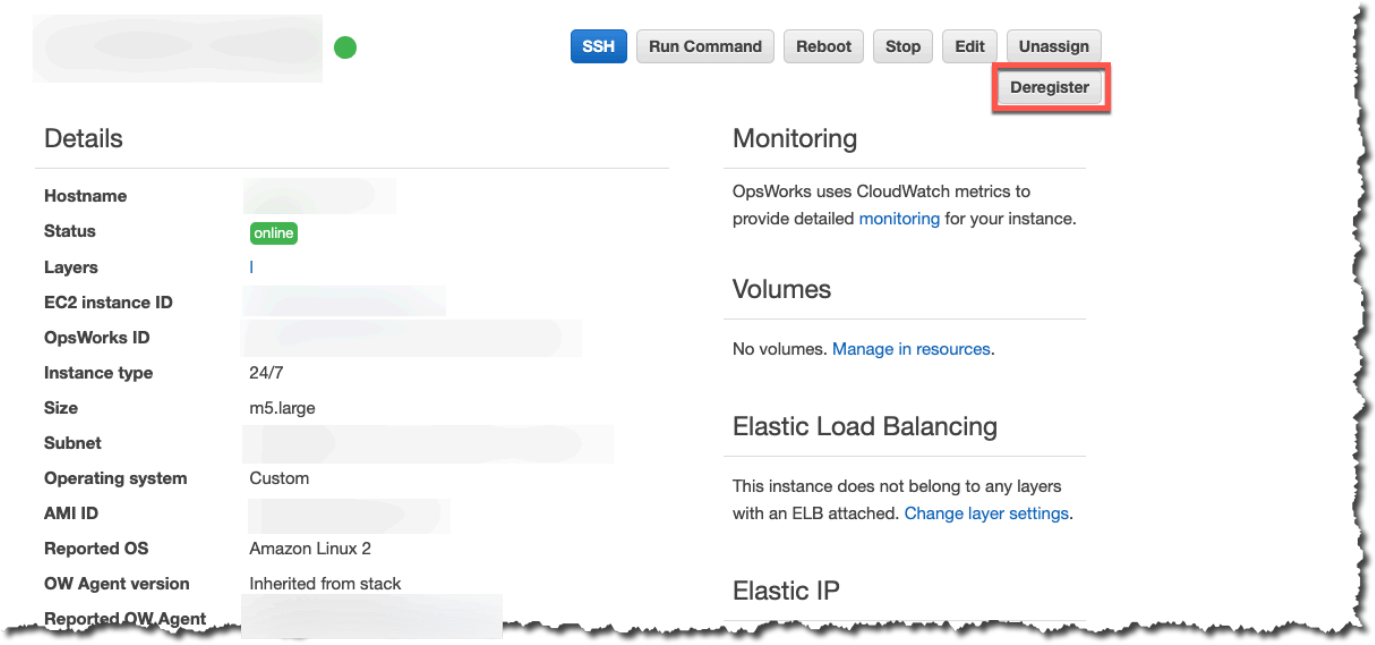
Deregistering a Registered Instance

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can deregister an instance using the OpsWorks console, AWS CLI, or SDK operation.

To deregister an instance using the console

1. In the navigation pane, choose **Instances**.
2. Choose the instance that you want to deregister.
3. On the **Details** page for the instance, choose **Deregister**.



To deregister an instance using the AWS CLI

Run the **aws opsworks deregister-instance** command to deregister an instance from its stack.

```
aws opsworks deregister-instance --region region --instance-id instance-id
```

When you deregister an instance, OpsWorks Stacks does the following:

- Removes the instance from the stack.
- Unassigns the instance from any assigned layers.
- Shuts down and uninstalls the agent.
- Deregisters any attached resources (Elastic IP addresses and Amazon EBS volumes).

  This procedure includes resources that were attached to the instance prior to registration, and resources that you used OpsWorks Stacks to attach to the instance while it was part of the stack.

After deregistration, the resources are no longer part of the stack's resources, but they remain attached to the instance.

- For on-premises instances, stops the charges.
- Removes all tags that OpsWorks added to the instance.

The instance remains in the running state, but it is under your direct control and is no longer managed by OpsWorks Stacks.

Note

Both registering and deregistering computers or instances are fully supported only within Linux stacks. For Windows stacks, deregistering instances is allowed, but it doesn't uninstall the OpsWorks agent from the instance. Deregistration does not remove all changed files, and does not fully revert to backed-up copies of certain files. This list applies to both Chef 11.10 and Chef 12 stacks; differences between the two versions are noted here.

- `/etc/hosts` is backed up to `/var/lib/aws/opsworks/local-mode-cache/backup/etc/`, but is not restored.
- Entries remain for `aws` and `opsworks` in passwd, group, and shadow files, etc.
- `/etc/sudoers` contains a reference to an OpsWorks Stacks directory.
- The following files are safe to leave behind; long-term, consider deleting `/var/lib/aws/opsworks`.
  - `/var/log/aws/opsworks` remains on instances in Chef 11.10 stacks.
  - `/var/lib/aws/opsworks` remains on both Chef 11.10 and Chef 12 stacks.
  - `/var/chef` remains on instances in Chef 12 stacks.
- Other files left behind:
  - `/etc/logrotate.d/opsworks-agent`
  - `/etc/cron.d/opsworks-agent-updater`
  - `/etc/ld.so.conf.d/opsworks-user-space.conf`
  - `/etc/motd.opsworks-static`
  - `/etc/aws/opsworks`
  - `/etc/sudoers.d/opsworks`
  - `/etc/sudoers.d/opsworks-agent`

Registered Instance Life Cycle

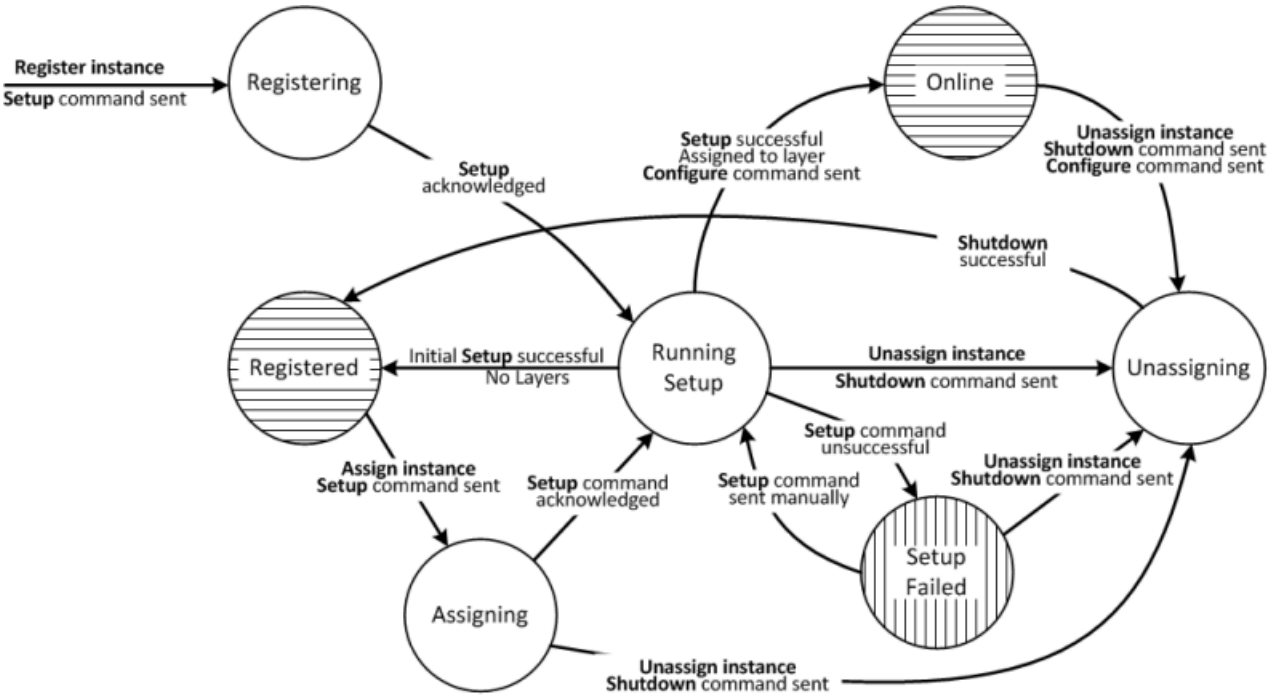
Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

This feature is supported only for Linux stacks.

The registered instance lifecycle starts after the agent is installed and running. At that point, it directs OpsWorks Stacks to register the instance with the stack. The following state diagram summarizes the key lifecycle elements.



Each state corresponds to an instance status. The edges represent one of the following OpsWorks Stacks commands. The details are discussed in the following sections.

- **Setup** – This command corresponds to the Setup lifecycle event and runs the instance's Setup recipes.
- **Configure** – This command corresponds to the Configure lifecycle event.

  OpsWorks Stacks triggers this event on every instance in the stack when an instance enters or leaves the online state. The instances run their Configure recipes, which make any changes that are required to accommodate the new instance.
- **Shutdown** – This command corresponds to the Shutdown lifecycle event, which runs the instance's Shutdown recipes.

  These recipes perform tasks such as shutting down services, but they do not stop the instance.
- **Deregister** – This command deregisters an instance and does not correspond to a lifecycle event.

> **Note**
>
> For simplicity the diagram does not show the Deregistering and Deleted states. You can deregister an instance from any of the states in the diagram, which sends a Deregister command to the instance and moves it to the Deregistering state.
>
> - If you deregister an online instance, OpsWorks Stacks sends a Configure command to the remaining instances in the stack to notify them that the instance is going offline.
> - After the Deregister command is acknowledged, the instance is still running, but it is in the Deleted state and no longer part of the stack. If you want to incorporate the instance into the stack again, you must re-register it.

Topics

- Registering
- Running Setup
- Registered
- Assigning
- Online
- Setup Failed
- Unassigning

- Initial Setup Configuration Changes

Registering

After the agent sends a registration request, OpsWorks Stacks starts the instance lifecycle by sending a Setup command to the instance, putting it in the Registering state. After the instance acknowledges the Setup command, it moves to the Running Setup state.

Running Setup

The Running Setup state runs the instance's Setup recipes. Setup works depending on the preceding state.

> **Note**
>
> If you unassign the instance while it is in the Running Setup state, OpsWorks Stacks sends a Shutdown command, which runs the instance's Shutdown recipes but does not stop the instance. The instance moves to the Unassigning state.

Topics

- Registering
- Assigning
- Setup Failed

Registering

During the Registering process, setup creates an OpsWorks Stacks instance to represent the registered instance in the stack, and runs a set of core Setup recipes on the instance.

One key change performed by initial setup is overwriting the instance's hosts file. By registering the instance, you have handed user management over to OpsWorks Stacks, which must have its own hosts file to control SSH login permissions. Initial setup also creates or modifies a number of files and, on Ubuntu systems, modifies package sources and installs a set of packages. For details, see Initial Setup Configuration Changes.

During registering, the process calls the IAM `AttachUserPolicy` that is part of the permissions attached to the IAM user that you create as a prerequisite. If `AttachUserPolicy` does not exist

(most likely because you are running an older release of the AWS CLI), the process falls back to calling `PutUserPolicy`.

> **Note**
>
> For consistency, OpsWorks Stacks runs every core Setup recipe. However, some of them perform some or all of their tasks only if an instance has been assigned to at least one layer, so they do not necessarily affect initial setup.

- If setup is successful, the instance moves to the Registered state.
- If setup is unsuccessful, the instance moves to the Setup Failed state.

Assigning

The instance has at least one assigned layer. OpsWorks Stacks runs each layer's Setup recipes, including any custom recipes that you have assigned to the layers' Setup event.

- If setup is successful, the instance moves to the Online state and OpsWorks Stacks triggers a Configure lifecycle event on every instance in the stack to notify them of the new instance.
- If setup is unsuccessful, the instance moves to the Setup Failed state.

> **Note**
>
> This setup process runs the core recipes a second time. However, Chef recipes are idempotent, so they do not repeat any tasks that have already been performed.

Setup Failed

If a setup process for an instance in the Assigning state fails, you can try again by using the Setup stack command to manually rerun the instance's Setup recipes.

- If setup is successful, the assigned instance moves to the Online state and OpsWorks Stacks triggers a Configure lifecycle event on every instance in the stack to notify them of the new instance.
- If the setup attempt is unsuccessful, the instance moves back to the Setup Failed state.

Registered

Instances in the Registered state are part of the stack and are managed by OpsWorks Stacks but are not assigned to a layer. They can remain in this state indefinitely.

If you assign the instance to one or more layers, OpsWorks Stacks sends a Setup command to the instance and it moves to the Assigning state.

Assigning

After the instance acknowledges the Setup command, it moves to the Running Setup state.

If you unassign the instance while it is in the Assigning state, OpsWorks Stacks terminates the setup process and sends a Shutdown command. The instance moves to the Unassigning state.

Online

The instance is now a member of at least one layer and is treated like a regular OpsWorks Stacks instance. It can remain in this state indefinitely.

If you unassign the instance while it is in the Online state, OpsWorks Stacks sends a Shutdown command to the instance and a Configure command to the rest of the stack's instances. The instance moves to the Unassigning state.

Setup Failed

The Setup command has failed.

- You can try again by running the Setup stack command.

  The instance returns to the Running Setup state.
- If you unassign the instance, OpsWorks Stacks sends a Shutdown command to the instance.

  The instance moves to the Unassigning state.

Unassigning

After the Shutdown command finishes, the instance is no longer assigned to any layers and returns to the Registered state.

> **Note**
>
> If an instance is assigned to multiple layers, unassignment applies to every layer; you cannot unassign a subset of the assigned layers. If you want a different set of assigned layers, unassign the instance and then reassign the desired layers.

Initial Setup Configuration Changes

Initial setup creates or modifies the following files and directories on all registered instances.

Created Files

```
/etc/apt/apt.conf.d/99-no-pipelining
/etc/aws/
/etc/init.d/opsworks-agent
/etc/motd
/etc/motd.opsworks-static
/etc/sudoers.d/opsworks
/etc/sudoers.d/opsworks-agent
/etc/sysctl.d/70-opsworks-defaults.conf
/opt/aws/opsworks/
/usr/sbin/opsworks-agent-cli
/var/lib/aws/
/var/log/aws/
/vol/
```

Modified Files

```
/etc/apt/apt.conf.d/99-no-pipelining
/etc/crontab
/etc/default/monit
/etc/group
/etc/gshadow
/etc/monit/monitrc
/etc/passwd
/etc/security/limits.conf (removing limits only for EC2 micro instances)
/etc/shadow
/etc/sudoers
```

Initial setup also creates a swap file on Amazon EC2 micro instances.

Initial setup makes the following changes to Ubuntu systems.

Package Sources

Initial setup changes package sources to the following.

- `deb http://archive.ubuntu.com/ubuntu/ ${code_name} main universe`

  To: `deb-src http://archive.ubuntu.com/ubuntu/ ${code_name} main universe`
- `deb http://archive.ubuntu.com/ubuntu/ ${code_name}-updates main universe`

  To: `deb-src http://archive.ubuntu.com/ubuntu/ ${code_name}-updates main universe`
- `deb http://archive.ubuntu.com/ubuntu ${code_name}-security main universe`

  To: `deb-src http://archive.ubuntu.com/ubuntu ${code_name}-security main universe`
- `deb http://archive.ubuntu.com/ubuntu/ ${code_name}-updates multiverse`

  To: `deb-src http://archive.ubuntu.com/ubuntu/ ${code_name}-updates multiverse`
- `deb http://archive.ubuntu.com/ubuntu ${code_name}-security multiverse`

  To: `deb-src http://archive.ubuntu.com/ubuntu ${code_name}-security multiverse`
- `deb http://archive.ubuntu.com/ubuntu/ ${code_name} multiverse`

  To: `deb-src http://archive.ubuntu.com/ubuntu/ ${code_name} multiverse`
- `deb http://security.ubuntu.com/ubuntu ${code_name}-security multiverse`

  To: `deb-src http://security.ubuntu.com/ubuntu ${code_name}-security multiverse`

Packages

Initial setup uninstalls `landscape` and installs the following packages.

`autofs`	`libicu-dev`	`libopenssl-ruby`

libssl-dev	libxml2-dev	libxslt-dev
libyaml-dev	monit	ntpd
procps	ruby	ruby-dev
rubygems	screen	sqlite
vim	xfs	

Editing the Instance Configuration

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can edit instance configurations, including registered Amazon Elastic Compute Cloud (Amazon EC2) instances, with the following limitations:

- The instance must be in the stopped state.

  Although you can't modify an online instance's properties, you can change some aspects of its configuration by editing the instance's layers. For more information, see Editing an OpsWorks Layer's Configuration.
- Some settings, such as **Availability Zone** and **Scaling Type**, are determined when you create the instance and cannot be modified later.
- Some settings can be modified for instance store-backed instances only, not for Amazon Elastic Block Store-backed instances.

  For example, you can change an instance store-backed instance's operating system. Amazon EBS-backed instances must use the operating system that you specified when you created the instance. For more information on instance storage, see Storage.

- By default, instances inherit the stack's agent version setting.

  You can use **OpsWorks Agent Version** to override the stack's agent version setting and specify a particular agent version for an instance. If you specify an instance's agent version, OpsWorks Stacks does not automatically update the agent when a new version is available, even if the stack's agent version setting is **Auto-update**. You must update the instance's agent version manually by editing the instance configuration. OpsWorks Stacks then installs the specified agent version on the instance.

> **Note**
>
> You cannot edit the configuration of registered on-premises instances.

To edit an instance's configuration

1. Stop the instance, if it is not already stopped.
2. On the **Instances** page, click an instance name to display the **Details** page.
3. Click **Edit** to display the edit page.
4. Edit the instance's configuration, as appropriate.

For a description of the **Host name**, **Size**, **SSH key** and **Operating system** settings, see Adding an Instance to a Layer. The **Layers** setting lets you add or remove layers. The instance's current layer's appear following the list of layers.

- To add another layer, select it from the list.
- To remove the instance from one of its layers, click the **x** by the appropriate layer.

  An instance must be a member of at least one layer, so you cannot remove the last layer.

When you restart the instance, OpsWorks Stacks starts a new Amazon EC2 instance with the updated configuration.

Deleting OpsWorks Stacks Instances

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can use OpsWorks Stacks to stop an instance, including registered Amazon EC2 instances. Doing so stops the EC2 instance, but the instance remains in the stack. You can restart it by clicking **start** in the instance's **Actions** column. If you no longer need an instance and want to remove it from the stack, you can delete it, which removes the instance from the stack and terminates the associated Amazon EC2 instance. Deleting an instance also deletes any associated logs or data, and any Amazon Elastic Block Store (EBS) volumes on the instance.

Important

This topic applies only to Amazon EC2 instances that are managed by OpsWorks Stacks. For more information about how to delete instances that are managed by the Amazon EC2 console or API, see Terminate Your Instance.

Note

You cannot use OpsWorks Stacks to delete a registered on-premises instance.

If an instance belongs to multiple layers, you can delete the instance from the stack or just remove a particular layer. You can also remove layers from instances by editing the instance configuration, as described in Editing the Instance Configuration.

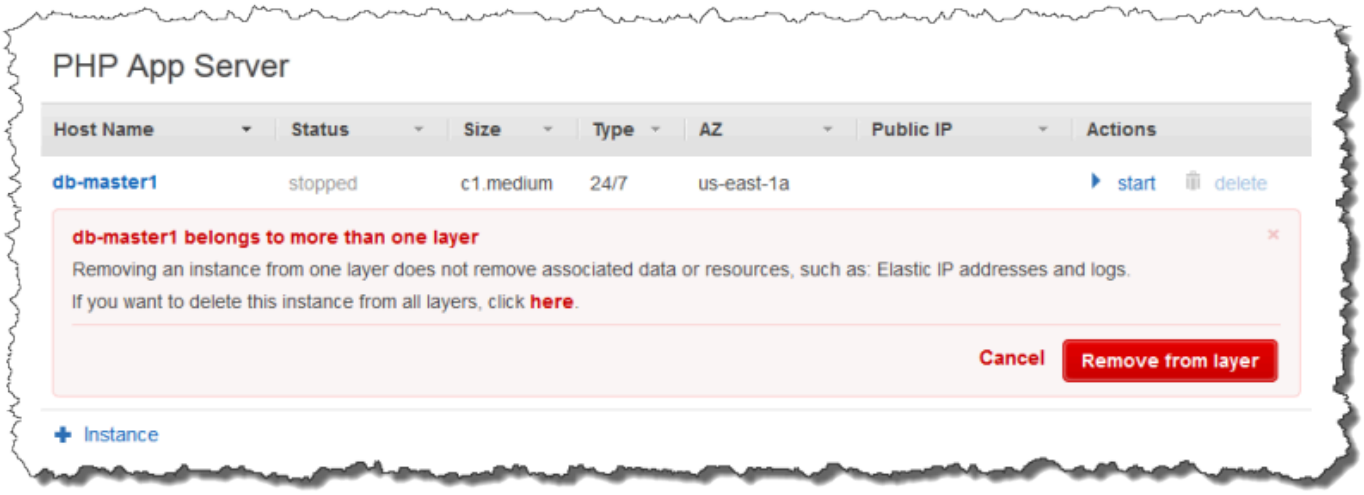
Important

You should delete OpsWorks Stacks instances only by using the OpsWorks Stacks console or API. In particular, you should not delete OpsWorks Stacks instances by using the Amazon

EC2 console or API because Amazon EC2 actions are not automatically synchronized with OpsWorks Stacks. For example, if auto healing is enabled and you terminate an instance by using the Amazon EC2 console, OpsWorks Stacks treats the terminated instance as a failed instance and launches another Amazon EC2 instance to replace it. For more information, see Using Auto Healing.

To delete an instance

1. On the **Instances** page, locate the instance under the appropriate layer. If the instance is running, click **stop** in the **Actions** column.
2. After the status changes to **stopped**, click **delete**. If the instance is a member of more than one layer, layer OpsWorks Stacks displays the following section.



   - To remove the instance from only the selected layer, click **Remove from layer**.

     The instance remains a member of its other layers and can be restarted.
   - To delete the instance from all its layers, which removes it from the stack, click **here**.
3. If you choose to completely remove an instance from the stack, or if the instance is a member of only one layer, OpsWorks Stacks prompts you to confirm the deletion.

   Choose **Delete** to confirm. In addition to deleting the instance from the stack, this action deletes any associated logs or data, and root volumes that are attached to the instance. To remove all instance volumes, choose **Delete instance's EBS volumes (snapshots will not be deleted)** before you choose **Delete**.

Using SSH to Log In to a Linux Instance

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can log into your online Linux instances with SSH using either the built-in MindTerm client, or a third-party client, such as PuTTY. SSH typically depends on an RSA key pair for authentication. You install the public key on the instance and provide the corresponding private key to the SSH client. OpsWorks Stacks handles installing public keys on your stack's instances for you, as follows.

- Amazon Elastic Compute Cloud (Amazon EC2)key pair – If the stack's region has one or more Amazon EC2 key pairs, you can specify a default SSH key pair for the stack.

  You can optionally override the default key pair and specify a different pair when you create an instance. In either case, OpsWorks Stacks installs the specified key pair's public key on the instance. For more information on how to create Amazon EC2 key pairs, see Amazon EC2 Key Pairs.
- Personal key pair – Each user can register a personal key pair with OpsWorks Stacks.

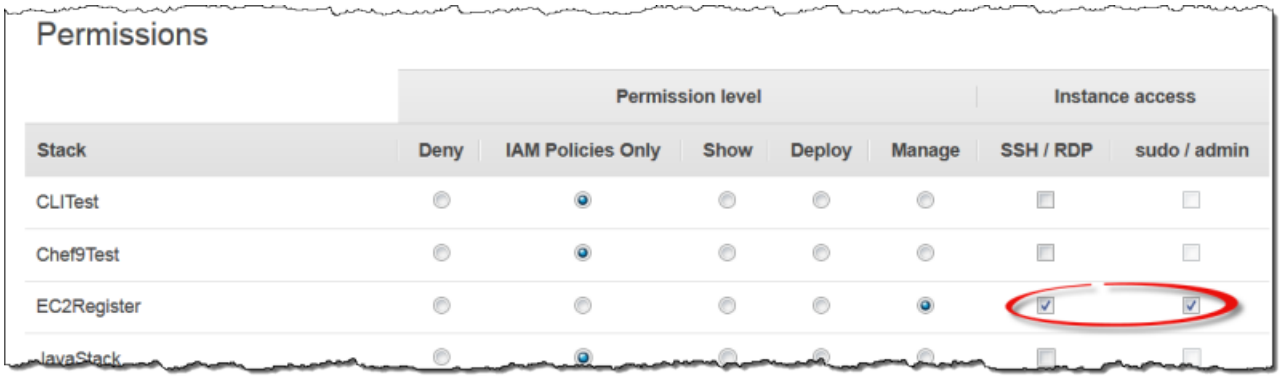
  The user or an administrator registers the public key with OpsWorks Stacks, and the user stores the private key locally. When setting permissions for a stack, the administrator specifies which users should have SSH access to the stack's instances. OpsWorks Stacks automatically creates a system user on the stack's instances for each authorized user and installs the users' personal public key.

A user must have SSH authorization to use the MindTerm SSH client or to use their personal key pair to log in to a stack's instances.

To authorize SSH for a user

1. In the OpsWorks Stacks navigation pane, click **Permissions**.

2. Select **SSH/RDP** for the desired IAM user to grant the necessary permissions. If you want to allow the user to use sudo to elevate privileges—for example, to run agent CLI commands—select **sudo/admin** also.



For more information on how to use OpsWorks Stacks to manage SSH access, see Managing SSH Access.

Topics

- Using the Built-in MindTerm SSH Client
- Using a Third-Party SSH Client

Using the Built-in MindTerm SSH Client

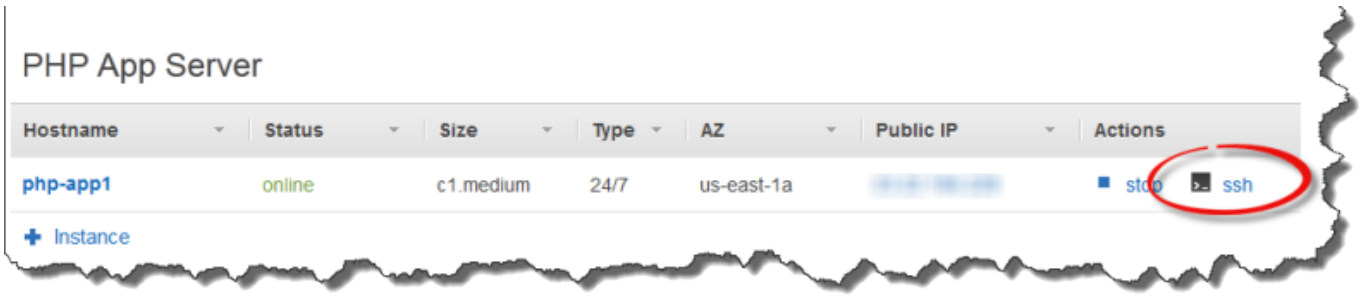
> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The simplest way to log into a Linux instance is to use the built-in MindTerm SSH client. Each online instance includes an **ssh** action that you can use to launch the MindTerm client.

> **Note**
>
> You must have Java enabled in your browser to use the MindTerm client.

To log in with the MindTerm client

1. If you haven't done so already, authorize SSH access for the IAM user that will be connecting to the instance, as described in the preceding section.
2. Log in as the user.
3. On the **Instances** page, choose **ssh** in the **Actions** column for the appropriate instance.



4. For **Private key**, provide a path to the user's personal private key or an Amazon EC2 private key, depending on which public keys you have installed on the instance.
5. Choose **Launch Mindterm** and use the terminal window to run commands on the instance.

Using a Third-Party SSH Client

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can also use a third-party SSH client, such as PuTTY, to connect to Linux instances.

To use a third party SSH client

1. Ensure that OpsWorks Stacks has installed an Amazon EC2 public key or an IAM user's personal public key on the instance, as discussed earlier.
2. Obtain the instance's public DNS name or public IP address from its details page.

3. Provide the client with the instance's host name, which depends on the operating system, as follows:

   - Amazon Linux and Red Hat Enterprise Linux (RHEL)– `ec2-user@`*`DNSName/Address`*`.`
   - Ubuntu – `ubuntu@`*`DNSName/Address`*.

   Replace *`DNSName/Address`* with the public DNS name or IP address from the previous step.

4. Provide the client with a private key that corresponds to an installed public key. You can use either an Amazon EC2 private key or an IAM user's personal private key, depending on which public keys have been installed on the instance.

Using RDP to Log In to a Windows Instance

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can use the Windows remote desktop protocol (RDP) to log in to an online Windows instance, as follows:

- The instance must have a security group with an inbound rule that allows RDP access.

  For more information on working with security groups, see Using Security Groups.

- Ordinary users – OpsWorks Stacks provides authorized ordinary users with an RDP password that is valid for a limited time period, which can range from 30 minutes to 12 hours.

  In addition to being authorized, users must have at least a Show permission level or their attached AWS Identity and Access Management (IAM) policies must allow the `opsworks:GrantAccess` action.

- Administrators – You can use the Administrator password to log in for an unlimited amount of time.

As described later, if you have specified an Amazon Elastic Compute Cloud (Amazon EC2) key pair for the instance, you can use it to retrieve the Administrator password.

> **Note**
>
> This topic describes how to use the Windows Remote Desktop Connection client to log in from a Windows workstation. You can also use one of the available RDP clients for Linux or OS X, but the procedure might be somewhat different. For more information on RDP clients that are compatible with Microsoft Windows Server 2012 R2, see Microsoft Remote Desktop Clients.

Topics

- Providing a Security Group that Allows RDP Access
- Logging in As an Ordinary User
- Logging in As Administrator

Providing a Security Group that Allows RDP Access

Before you can use RDP to log into a Windows instance, the instance's security group inbound rules must allow RDP connections. When you create the first stack in a region, OpsWorks Stacks creates a set of security groups. They include one named something like `AWS-OpsWorks-RDP-Server`, which OpsWorks Stacks attaches to all Windows instances to allow RDP access. However, by default, this security group does not have any rules, so you must add an inbound rule to allow RDP access to your instances.

To allow RDP access

1. Open the Amazon EC2 console, set it to the stack's region, and choose **Security Groups** from the navigation pane.
2. Select **AWS-OpsWorks-RDP-Server**, choose the **Inbound** tab, and choose **Edit**.
3. Choose **Add Rule** and specify the following settings:
   - **Type** – **RDP**
   - **Source** – The permissible source IP addresses.

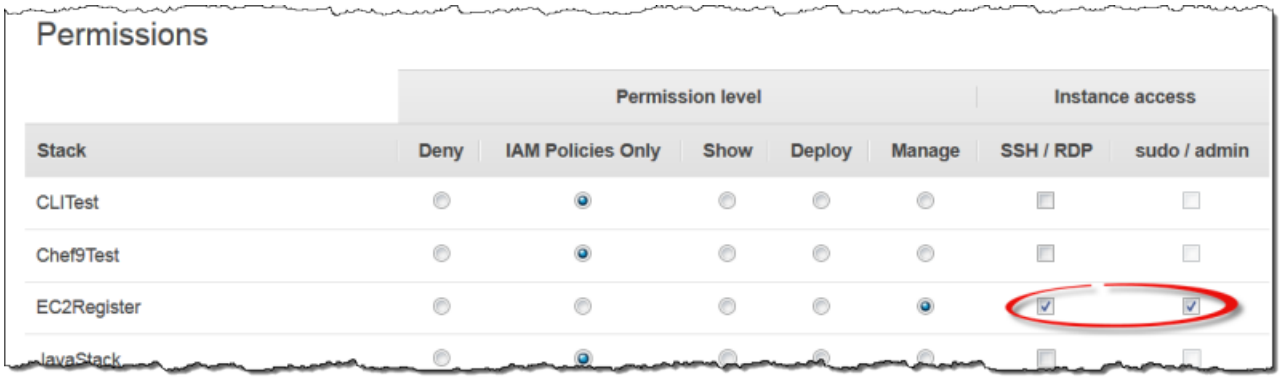
You typically allow inbound RDP requests from your IP address or a specified IP address range (typically your corporate IP address range).

Logging in As an Ordinary User

An authorized user can log in to instances using a temporary password, provided by OpsWorks Stacks.

To authorize RDP for a user;

1. In the OpsWorks Stacks navigation pane, click **Permissions**.
2. Select the **SSH/RDP** checkbox for the desired user to grant the necessary permissions. If you want the user to have administrator permissions, you should also select **sudo/admin**.



Authorized users can log in to any of the stack's online instances, as follows.

To log in as an ordinary IAM user

1. Log in as an IAM user.
2. On the **Instances** page, choose **rdp** in the **Actions** column for the appropriate instance.
3. Specify the session length, which can vary from 30 minutes to 12 hours, and choose **Generate Password**. The password will be valid only for the specified session duration.
4. Record the **public DNS name**, **username**, and **password** values, then choose **Acknowledge and close**.
5. Open the Windows Remote Desktop Connection client, choose **Show Options,** and provide the following from the information that you recorded in Step 4:

   - **Computer** – The instance's public DNS name.

You can also use the public IP address, if you prefer. Choose **Instances** and copy the address from the instance's **Public IP** column.

- **User name** – The user name.

6. When the client prompts for your credentials, enter the password that you saved in Step 4.

> **Note**
>
> OpsWorks Stacks generates a user password only for online instances. If you start an instance and, for example, one of your custom Setup recipes fails, the instance will end up in the `setup_failed` state. Even though the instance is not online as far as OpsWorks Stacks is concerned, the EC2 instance is running and it's often useful to log in to troubleshoot the issue. OpsWorks Stacks won't generate a password for you in this case, but if you have assigned an SSH key pair to the instance, you can use the EC2 console or CLI to retrieve the instance's Administrator password and log in as Administrator. For more information, see the following section.

Logging in As Administrator

You can log in to an instance as Administrator by using the appropriate password. If you have assigned an EC2 key pair to an instance, Amazon EC2 uses it to automatically create and encrypt an Administrator password when the instance starts. You can then use the key pair's private key with the EC2 console, API, or CLI to retrieve and decrypt the password.

> **Note**
>
> You cannot use a personal SSH key pair to retrieve an Administrator password; you must use an EC2 key pair.

The following describes how to use the EC2 console to retrieve an Administrator password and log in to an instance. If you prefer command-line tools, you can also use the AWS CLI `get-password-data` command to retrieve the password.

To log in as Administrator

1. Make sure that you have specified an EC2 key pair for the instance. You can specify a default key pair for all of the stack's instances when you create the stack, or you can specify a key pair for a particular instance when you create the instance.
2. Open the EC2 console, set it to the stack's region, and choose **Instances** from the navigation pane.
3. Select the instance, choose **Connect**, and choose **Get Password**.
4. Provide a path to the EC2 key pair's private key on your workstation, and choose **Decrypt Password**. Copy the decrypted password for later use.
5. Open the Windows Remote Desktop Connection client, choose **Show Options,** and provide the following information:
   - **Computer** – The instance's public DNS name or public IP address, which you can get from the instance's details page.
   - **User name** – `Administrator`.
6. When the client prompts for your credentials, provide the decrypted password from Step 4.

Apps

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

An OpsWorks Stacks *app* represents code that you want to run on an application server. The code itself resides in a repository such as an Amazon S3 archive; the app contains the information required to deploy the code to the appropriate application server instances.

When you deploy an application, OpsWorks Stacks triggers a Deploy event, which runs each layer's Deploy recipes. OpsWorks Stacks also installs stack configuration and deployment attributes that

contain all of the information needed to deploy the app, such as the app's repository and database connection data.

You must implement custom recipes that retrieve the app's deployment data from the stack configuration and deployment attributes and handle the deployment tasks.

Topics

- Adding Apps
- Deploying Apps
- Editing Apps
- Connecting an Application to a Database Server
- Using Environment Variables
- Passing Data to Applications
- Using Git Repository SSH Keys
- Using Custom Domains
- Using SSL

Adding Apps

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The first step in deploying an application to your application servers is to add an app to the stack. The app represents the application, and contains a variety of metadata, such as the application's name and type, and the information required to deploy the application to the server instances, such as the repository URL. You must have Manage permissions to add an app to a stack. For more information, see Managing User Permissions.

> **Note**
>
> The procedure in this section applies to Chef 12 and newer stacks. For information about how to add apps to layers in Chef 11 stacks, see Step 2.4: Create and Deploy an App - Chef 11.

To add an app to a stack

1. Put the code in your preferred repository—an Amazon S3 archive, a Git repository, a Subversion repository, or an HTTP archive. For more information, see Application Source.
2. Click **Apps** in the navigation pane. On the **Apps** page, click **Add an app** for your first app. For subsequent apps, click **+App**.
3. Use the **App New** page to configure the app, as described in the following section.

Configuring an App

The **Add App** page consists of the following sections: **Settings**, **Application source**, **Data Sources**, **Environment Variables**, **Add Domains**, and **SSL Settings**.

Topics

- Settings
- Application Source
- Data Sources
- Environment Variables
- Domain and SSL Settings

Settings

Name

The app name, which is used to represent the app in the UI. OpsWorks Stacks also uses this name to generate a short name for the app that is used internally and to identify the app in the stack configuration and deployment attributes. After you have added the app to the stack, you can see the short name by clicking **Apps** in the navigation pane and then clicking the app's name to open the details page.

Document root

OpsWorks Stacks assigns the **Document root** setting to the `[:document_root]` attribute in the app's `deploy` attributes. The default value is `null`. Your deployment recipes can obtain that value from the `deploy` attributes using standard Chef node syntax and deploy the specified code to the appropriate location on the server. For more information about how to deploy apps, see Deploy Recipes.

Application Source

You can deploy apps from the following repository types: Git, Amazon S3 bundle, HTTP bundle, and Other. All repository types require you to specify the repository type and the repository URL. Individual repository types have their own requirements, as explained below.

> **Note**
>
> OpsWorks Stacks automatically deploys applications from the standard repositories to the built-in server layers. If you use the Other repository type, which is the only option for Windows stacks, OpsWorks Stacks puts the repository information in the app's `deploy` attributes, but you must implement custom recipes to handle the deployment tasks.

Topics

- HTTP Archive
- Amazon S3 Archive
- Git Repository
- Other Repositories

HTTP Archive

To use a publicly-accessible HTTP server as a repository:

1. Create a compressed archive—zip, gzip, bzip2, Java WAR, or tarball—of the folder that contains the app's code and any associated files.

> **Note**
>
> OpsWorks Stacks does not support uncompressed tarballs.

2. Upload the archive file to the server.
3. To specify the repository in the console, select HTTP Archive as the repository type and enter the URL.

   If the archive is password-protected, under **Application Source**, specify the sign-in credentials.

Amazon S3 Archive

To use an Amazon Simple Storage Service bucket as a repository:

1. Create a public or private Amazon S3 bucket. For more information, see Amazon S3 Documentation.
2. For OpsWorks Stacks to access private buckets, you must be a user with at least read-only rights to the Amazon S3 bucket and you will need the access key ID and secret access key. For more information, see AWS Identity and Access Management Documentation.
3. Put the code and any associated files in a folder and store the folder in a compressed archive—zip, gzip, bzip2, Java WAR, or tarball.

   > **Note**
   >
   > OpsWorks Stacks does not support uncompressed tarballs.

4. Upload the archive file to the Amazon S3 bucket and record the URL.
5. To specify the repository in the OpsWorks Stacks console, set **Repository type** to **S3 Archive** and enter the archive's URL. For a private archive, you must also provide an AWS access key ID and secret access key whose policy grants permissions to access the bucket. Leave these settings blank for public archives.

Git Repository

A Git repository provides source control and versioning. OpsWorks Stacks supports publicly hosted repository sites such as GitHub or Bitbucket as well as privately hosted Git servers. For both apps

and Git submodules, the format you use to specify the repository's URL in **Application Source** depends on whether the repository is public or private:

Public repository–Use the HTTPS or Git read-only protocols. For example, Getting Started with Chef 11 Linux Stacks uses a public GitHub repository that can be accessed by either of the following URL formats:

- Git read-only: **`git://github.com/amazonwebservices/opsworks-demo-php-simple-app.git`**
- HTTPS: **`https://github.com/amazonwebservices/opsworks-demo-php-simple-app.git`**

Private repository–Use the SSH read/write format shown in these examples:

- Github repositories: **`git@github.com:`*`project`*`/`*`repository`*.
- Repositories on a Git server: *`user`*`@`*`server`*`:`*`project`*`/`*`repository`*

Selecting **Git** under **Source Control** displays two additional optional settings:

Repository SSH key

You must specify a deploy SSH key to access private Git repositories. This field requires the private key; the public key is assigned to your Git repository. For Git submodules, the specified key must have access to those submodules. For more information, see Using Git Repository SSH Keys.

> **Important**
>
> The deploy SSH key cannot require a password; OpsWorks Stacks has no way to pass it through.

Branch/Revision

If the repository has multiple branches, OpsWorks Stacks downloads the master branch by default. To specify a particular branch, enter the branch name, SHA1 hash, or tag name. To specify a particular commit, enter the full 40-hexdigit commit identifier.

Other Repositories

If the standard repositories do not meet your requirements, you can use other repositories, such as Bazaar. However, OpsWorks Stacks does not automatically deploy apps from such repositories. You must implement custom recipes to handle the deployment process and assign them to the appropriate layers' Deploy events. For an example of how to implement Deploy recipes, see Deploy Recipes.

Data Sources

This section attaches a database to the app. You have the following options:

- **RDS** – Attach one of the stack's Amazon RDS service layers.
- **None** – Do not attach a database server.

If you select **RDS**, you must specify the following.

Database instance

The list includes every Amazon RDS service layer. You can also select one of the following:

(Required) Specify which database server to attach to the app. The contents of the list depend on the data source.

- **RDS** – A list of the stack's Amazon RDS service layers.

Database name

(Optional) Specify a database name.

- Amazon RDS layer – Enter the database name that you specified for the Amazon RDS instance.

  You can get the database name from the Amazon RDS console.

When you deploy an app with an attached database, OpsWorks Stacks adds the database instance's connection to the app's `deploy` attributes.

You can write a custom recipe to retrieve the information from the `deploy` attributes and put it file that can be accessed by the application. This is the only option for providing database connection information to the Other application type.

For more information on how to handle database connections, see Connecting to a Database.

To detach a database server from an app, edit the app's configuration to specify a different database server, or no server.

Environment Variables

You can specify a set of environment variables for each app, which are specific to the app. For example, if you have two apps, the environment variables that you define for the first app are not available to the second app and vice versa. You can also define the same environment variable for multiple apps and assign it a different value for each app.

> **Note**
>
> There is no specific limit on the number of environment variables. However, the size of the associated data structure—which includes the variables' names, values, and protected flag values—cannot exceed 20 KB. This limit should accommodate most if not all use cases. Exceeding it will cause a service error (console) or exception (API) with the message, "Environment: is too large (maximum is 20 KB)."

OpsWorks Stacks stores the variables as attributes in the app's `deploy` attributes. You can have your custom recipes retrieve those values by using standard Chef node syntax. For examples of how to access an app's environment variables, see Using Environment Variables.

Key

The variable name. It can contain up to 64 upper and lower case letters, numbers, and underscores (_), but it must start with a letter or underscore.

Value

The variable's value. It can contain up to 256 characters, which must all be printable.

Protected value

Whether the value is protected. This setting allows you to conceal sensitive information such as passwords. If you set **Protected value** for a variable, after you create the app:

- The app's details page displays only the variable name, not the value.
- If you have permission to edit the app, you can click **Update value** to specify a new value, but you cannot see or edit the old value.

> **Note**
>
> Chef deployment logs can sometimes include environment variables. This means protected variables might be shown in the console. To prevent protected variables from being shown in the console, we recommend that you use Amazon S3 buckets as storage for protected variables that you do not want shown in the console. An example of how to use an S3 bucket for this purpose is available in Using an Amazon S3 Bucket in this guide.

Domain and SSL Settings

For the Other app type, OpsWorks Stacks adds the settings to the app's `deploy` attributes. Your recipes can retrieve the data from those attributes and configure the server as needed.

Domain Settings

This section has an optional **Add Domains** field for specifying domains. For more information, see Using Custom Domains.

SSL Settings

This section has an **SSL Support** toggle that you can use to enable or disable SSL. If you click **Yes**, you'll need to provide SSL certificate information. For more information, see Using SSL.

Deploying Apps

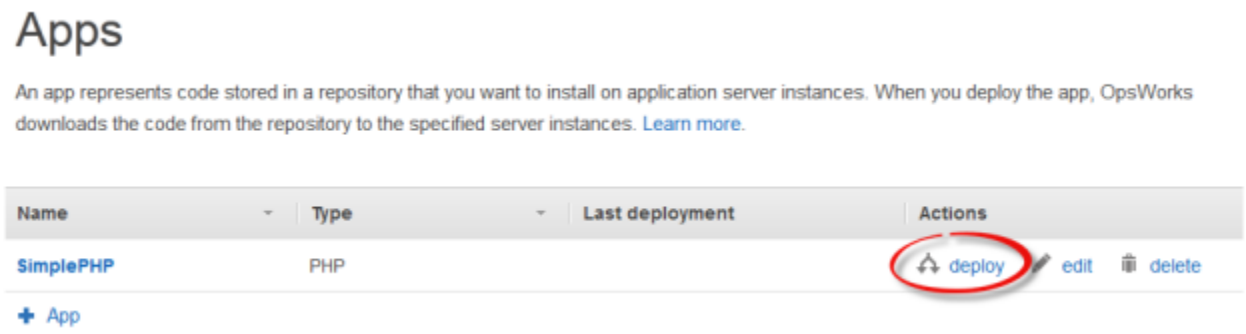
> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The primary purpose of deployment is to deploy application code and related files to application server instances. The deployment operation is handled by each instance's Deploy recipes, which are determined by the instance's layer.

When you start an instance, after the Setup recipes complete, OpsWorks Stacks automatically runs the instance's Deploy recipes. However, when you add or modify an app, you must deploy it manually to any online instances. You must have Manage or Deploy permissions to deploy an app. For more information, see Managing User Permissions.

To deploy an app

1. On the **Apps** page, click the app's **deploy** action.



> **Note**
>
> You can also deploy an app by clicking **Deployments** in the navigation pane. On the **Deployments & Commands** page, click **Deploy an app** When you do this, you can also choose which app to deploy.

2. Specify the following:

   - (Required) Set **Command:** to **deploy**, if it is not already selected.
   - (Optional) Include a comment.

3. Click **Advanced >>** to specify custom JSON. OpsWorks Stacks adds a set of stack configuration and deployment attributes to the node object. The `deploy` attributes contain the deployment details and can be used by Deploy recipes to handle installation and configuration. On Linux stacks, you can use the custom JSON field to override default OpsWorks Stacks settings or pass custom settings to your custom recipes. For more information about how to use custom JSON, see Using Custom JSON.

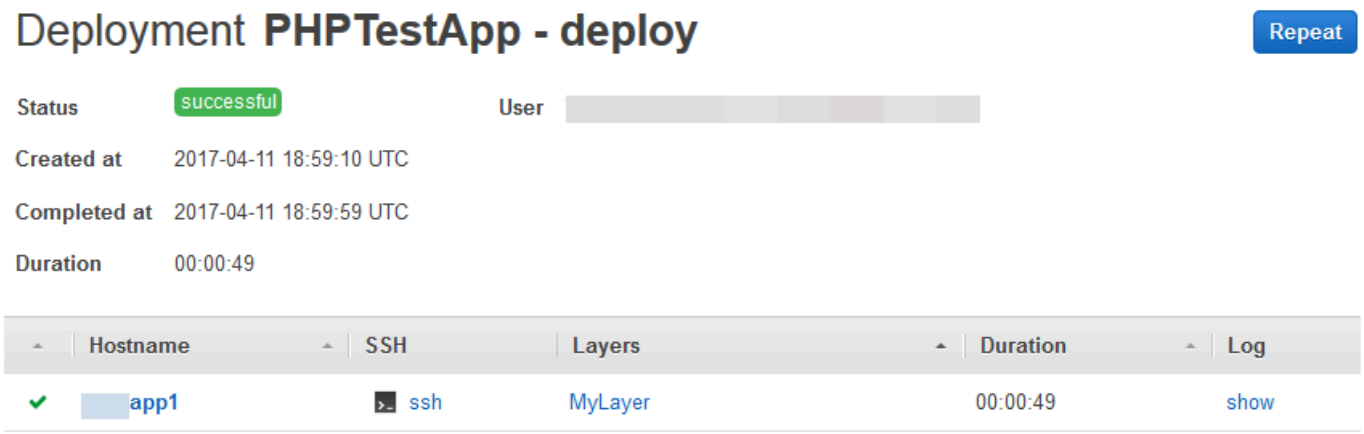
> **Note**
>
> If you specify custom JSON here, it is added to the stack configuration and deployment attributes for this deployment only. If you want to add custom JSON permanently, you

must add it to the stack. Custom JSON is limited to 120 KB. If you need more capacity, we recommend storing some of the data on Amazon S3. Your custom recipes can then use the AWS CLI or AWS SDK for Ruby to download the data from the bucket to your instance. For an example, see Using the SDK for Ruby.

4. Under **Instances**, click **Advanced >>** and specify which instances to run the deploy command on.

   The deploy command triggers a Deploy event, which runs the deploy recipes on the selected instances. The deploy recipes for the associated application server download the code and related files from the repository and install them on the instance, so you typically select all of the associated application server instances. However, other instance types might require some configuration changes to accommodate the new app, so it is often useful to run deploy recipes on those instances as well. Those recipes update the configuration as needed but do not install the app's files. For more information about recipes, see Cookbooks and Recipes.

5. Click **Deploy** to run the deploy recipes on the specified instances, which displays the Deployment page. When the process is complete, OpsWorks Stacks marks the app with a green check to indicate successful deployment. If deployment fails, OpsWorks Stacks marks the app with a red X. In that case, you can go to the **Deployments** page and examine the deployment log for more information.



Note

When you deploy an update to a JSP app, Tomcat might not recognize the update and instead continue to run the existing app version. This can happen, for example, if you deploy your app as a .zip file that contains only a JSP page. To ensure that Tomcat runs the most recently deployed version, the project's root directory should include a WEB-INF

directory that contains a `web.xml` file. A `web.xml` file can contain a variety of content, but the following is sufficient to ensure that Tomcat recognizes updates and runs the currently deployed app version. You don't have to change the version for each update. Tomcat will recognize the update even if the version hasn't changed.

```
<context-param>
  <param-name>appVersion</param-name>
  <param-value>0.1</param-value>
</context-param>
```

Other Deployment Commands

The **Deploy app** page includes several other commands for managing your apps and the associated servers. Of the following commands, only **Undeploy** is available for apps on Chef 12 stacks.

Undeploy

Triggers an Undeploy lifecycle event, which runs the undeploy recipes to remove all versions of the app from the specified instances.

Rollback

Restores the previously deployed app version. For example, if you have deployed the app three times and then run **Rollback**, the server will serve the app from the second deployment. If you run **Rollback** again, the server will serve the app from the first deployment. By default, OpsWorks Stacks stores the five most recent deployments, which allows you to roll back up to four versions. If you exceed the number of stored versions, the command fails and leaves the oldest version in place. This command is not available in Chef 12 stacks.

Start Web Server

Runs recipes that start the application server on the specified instances. This command is not available in Chef 12 stacks.

Stop Web Server

Runs recipes that stop the application server on the specified instances. This command is not available in Chef 12 stacks.

Restart Web Server

Runs recipes that restart the application server on the specified instances. This command is not available in Chef 12 stacks.

> **Important**
>
> **Start Web Server**, **Stop Web Server**, **Restart Web Server**, and **Rollback** are essentially customized versions of the Execute Recipes stack command. They run a set of recipes that perform the task on the specified instances.
>
> - These commands do not trigger a lifecycle event, so you cannot hook them to run custom code.
> - These commands work only for the built-in application server layers.
>
> In particular, these commands have no effect on custom layers, even if they support an application server. To start, stop, or restart servers on a custom layer, you must implement custom recipes to perform these tasks and use the Execute Recipes stack command to run them. For more information on how to implement and install custom recipes, see Cookbooks and Recipes.

Editing Apps

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can modify an app's configuration by editing the app. For example, if you are ready to deploy a new version, you can edit the app's OpsWorks Stacks settings to use the new repository branch. You must have Manage or Deploy permissions to edit an app's configuration. For more information, see Managing User Permissions.

To edit an app

1. On the **Apps** page click the app name to open its details page.
2. Click **Edit** to change the app's configuration.
   - If you modify the app's name, OpsWorks Stacks uses the new name to identify the app in the console.

     Changing the name does not change the associated short name. The short name is set when you add the app to the stack and cannot be subsequently modified.
   - If you have specified a protected environment variable, you cannot see or edit the value. However, you can specify a new value by clicking **Update value**.
3. Click **Save** to save the new configuration and then **Deploy App** to deploy the app.

Editing an app changes the settings with OpsWorks Stacks, but does not affect the stack's instances. When you first deploy an app, the Deploy recipes download the code and related files to the app server instances, which then run the local copy. If you modify the app in the repository or change any other settings, you must deploy the app to install the updates on your app server instances, as follows. OpsWorks Stacks automatically deploys the current app version to new instances when they are started. For existing instances, however, the situation is different:

- OpsWorks Stacks automatically deploys the current app version to new instances when they are started.
- OpsWorks Stacks automatically deploys the latest app version to offline instances, including load-based and time-based instances, when they are restarted.
- You must manually deploy the updated app to online instances.

For more information on how to deploy apps, see Deploying Apps

Connecting an Application to a Database Server

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can associate an Amazon RDS database server with an app when you create the app or later by editing the app. Your application can then use the database connection information—user name, password, ...—to connect to the database server. When you deploy an app, OpsWorks Stacks provides this information to applications in two ways:

- For Linux stacks, OpsWorks Stacks creates a file on each of the built-in application server instances containing the connection data that the application can use to connect to the database server.
- OpsWorks Stacks includes the connection information in the stack configuration and deployment attributes that are installed on each instance.

  You can implement a custom recipe to extract the connection information from these attributes and put it in a file in your preferred format. For more information, see Passing Data to Applications.

Important

For Linux stacks, if you want to associate an Amazon RDS service layer with your app, you must add the appropriate driver package to the associated app server layer, as follows:

1. Click **Layers** in the navigation pane and open the app server's **Recipes** tab.
2. Click **Edit** and add the appropriate driver package to **OS Packages**. For example, you should specify `mysql` if the layer contains Amazon Linux instances and `mysql-client` if the layer contains Ubuntu instances.
3. Save the changes and redeploy the app.

Using a Custom Recipe

You can implement a custom recipe that extracts the connection data from the app's `deploy` attributes and saves it in a form that the application can read, such as a YAML file.

You attach a database server to an app when you create the app or later by editing the app. When you deploy the app, OpsWorks Stacks installs a stack configuration and deployment attributes on each instance that include the database connection information. Your app can then retrieve the appropriate attributes. The details depend on whether you are using a Linux or Windows stack.

Connecting to a Database Server for a Linux Stack

For Linux stacks, the stack configuration and deployment attributes' `deploy` namespace includes an attribute for each deployed app, named with the app's short name. When you attach a database server to an app, OpsWorks Stacks populates the app's `[:database]` attribute with the connection information, and installs it on the stack's instances for each subsequent deployment. The attribute values are either user-provided or generated by OpsWorks Stacks.

> **Note**
>
> OpsWorks Stacks allows you to attach a database server to multiple apps, but each app can have only one attached database server. If you want to connect an application to more than one database server, attach one of the servers to the app, and use the information in the app's `deploy` attributes to connect to that server. Use custom JSON to pass the connection information for the other database servers to the application. For more information, see Passing Data to Applications.

An application can use the connection information from the instance's `deploy` attributes to connect to a database. However, applications cannot access that information directly—only recipes can access the `deploy` attributes. You can address this issue by implementing a custom recipe that extracts the connection information from the `deploy` attributes and puts it in a file that can be read by the application.

Using Environment Variables

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> The recommendations in this topic apply to Chef 11.10 and earlier versions of Chef. To get environment variables in Chef 12 and newer releases, you must use the App Data Bag. For more information, see AWS OpsWorks Data Bag Reference and App Data Bag (aws_opsworks_app).

When you specify environment variables for an app, OpsWorks Stacks adds the variable definitions to the app's `deploy` attributes.

Custom layers can use a recipe to retrieve a variable's value by using standard node syntax, and store it in a form that is accessible to the layer's apps.

You must implement a custom recipe that obtains the environment variable values from the instance's `deploy` attributes. The recipe can then store the data on the instance in a form that can be accessed by the application, such as a YAML file. An app's environment variable definitions are stored in the `deploy` attributes, in the app's `environment_variables`. The following example shows the location of these attributes for an app named `simplephpapp`, using JSON to represent the attribute structure.

```
{
  ...
  "ssh_users": {
  },
  "deploy": {
    "simplephpapp": {
      "application": "simplephpapp",
      "application_type": "php",
      "environment_variables": {
        "USER_ID": "168424",
        "USER_KEY": "somepassword"
      },
    ...
  }
}
```

A recipe can obtain variable values by using standard node syntax. The following example shows how to obtain the `USER_ID` value from the preceding JSON and place it in the Chef log.

```
Chef::Log.info("USER_ID: #{node[:deploy]['simplephpapp'][:environment_variables]
[:USER_ID]}")
```

For a more detailed description of how to retrieve information from the stack configuration and deployment JSON and store it on the instance, see Passing Data to Applications.

Passing Data to Applications

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

It is often useful to pass data such as key-value pairs to an application on the server. To do so, use custom JSON to add the data to the stack. OpsWorks Stacks adds the data to each instance's node object for each lifecycle event.

Note, however, that although recipes can get the custom JSON data from the node object by using Chef attributes, applications cannot. One approach to getting custom JSON data to one or more applications is to implement a custom recipe that extracts the data from the node object and writes it to a file that the application can read. The example in this topic shows how to write the data to a YAML file, but you can use the same basic approach for other formats, such as JSON or XML.

To pass key-value data to the stack's instances, add custom JSON like the following to the stack. For more information about how to add custom JSON to a stack, see Using Custom JSON.

```
{
  "my_app_data": {
    "app1": {
      "key1": "value1",
      "key2": "value2",
      "key3": "value3"
```

```
    },
    "app2": {
      "key1": "value1",
      "key2": "value2",
      "key3": "value3"
    }
  }
}
```

The example assumes that you have two apps whose short names are `app1` and `app2`, each of which has three data values. The accompanying recipe assumes that you use the apps' short names to identify the associated data; the other names are arbitrary. For more information on app short names, see Settings.

The recipe in the following example shows how to extract the data for each app from the `deploy` attributes and put it in a `.yml` file. The recipe assumes that your custom JSON contains data for each app.

```
node[:deploy].each do |app, deploy|
  file File.join(deploy[:deploy_to], 'shared', 'config', 'app_data.yml') do
    content YAML.dump(node[:my_app_data][app].to_hash)
  end
end
```

The `deploy` attributes contain an attribute for each app, named with the app's short name. Each app attribute contains a set of attributes that represent a variety of information about the app. This example uses the app's deployment directory, which is represented by the `[:deploy][:app_short_name][:deploy_to]` attribute. For more information on `[:deploy]`, see deploy Attributes.

For each app in `deploy`, the recipe does the following:

1. Creates a file named `app_data.yml` in the `shared/config` subdirectory of the application's `[:deploy_to]`directory.

   For more information on how OpsWorks Stacks installs apps, see Deploy Recipes.
2. Converts the app's custom JSON values to YAML and writes the formatted data to `app_data.yml`.

To pass data to an app

1. Add an app to the stack and note its short name. For more information, see Adding Apps.
2. Add custom JSON with the app's data to the `deploy` attributes, as described earlier. For more information on how to add custom JSON to a stack, see Using Custom JSON.
3. Create a cookbook and add a recipe to it with code based on the previous example, modified as needed for the attribute names that you used in the custom JSON. For more information on how to create cookbooks and recipes, see Cookbooks and Recipes. If you already have custom cookbooks for this stack, you could also add the recipe to an existing cookbook, or even add the code to an existing Deploy recipe.
4. Install the cookbook on your stack. For more information, see Installing Custom Cookbooks.
5. Assign the recipe to the app server layer's Deploy lifecycle event. OpsWorks Stacks will then run the recipe on each new instance, after it has booted. For more information, see Executing Recipes.
6. Deploy the app, which also installs stack configuration and deployment attributes that now contain your data.

Note

If the data files must be in place before the app is deployed, you can also assign the recipe to the layer's Setup lifecycle event, which occurs once, right after the instance finishes booting. However, OpsWorks Stacks will not have created the deployment directories yet, so your recipe should create the required directories explicitly prior to creating the data file. The following example explicitly creates the app's `/shared/config` directory, and then creates a data file in that directory.

```
node[:deploy].each do |app, deploy|

 directory "#{deploy[:deploy_to]}/shared/config" do
      owner "deploy"
      group "www-data"
      mode 0774
      recursive true
      action :create
    end
```

```
  file File.join(deploy[:deploy_to], 'shared', 'config', 'app_data.yml') do
    content YAML.dump(node[:my_app_data][app].to_hash)
  end
end
```

To load the data, you can use something like the following Sinatra code:

```
#!/usr/bin/env ruby
# encoding: UTF-8
require 'sinatra'
require 'yaml'

get '/' do
  YAML.load(File.read(File.join('..', '..', 'shared', 'config', 'app_data.yml')))
End
```

You can update the app's data values at any time by updating the custom JSON, as follows.

To update the app data

1. Edit the custom JSON to update the data values.
2. Deploy the app again, which directs OpsWorks Stacks to run the Deploy recipes on the stack's instances. The recipes will use attributes from the updated stack configuration and deployment attributes, so your custom recipe will update the data files with the current values.

Using Git Repository SSH Keys

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

A Git repository SSH key, sometimes called a deploy SSH key, is an SSH key with no password that provides access to a private Git repository. Ideally, it doesn't belong to any specific developer. Its purpose is to allow OpsWorks Stacks to asynchronously deploy apps or cookbooks from a Git repository without requiring any further input from you.

The following describes the basic procedure for creating a repository SSH key. For details, see the documentation for your repository. For example, Managing deploy keys describes how to create a repository SSH key for a GitHub repository, and Deployment Keys on Bitbucket describes how to create a repository SSH key for a Bitbucket repository. Note that some documentation describes creating a key on a server. For OpsWorks Stacks, just replace "server" with "workstation" in the instructions.

To create a repository SSH key

1. Create a deploy SSH key pair for your Git repository on your workstation using a program such as `ssh-keygen`.

   > **Important**
   >
   > OpsWorks Stacks does not support SSH key passphrases.

2. Assign the public key to the repository and store the private key on your workstation.
3. Enter the private key in the **Repository SSH Key** box when you add an app or specify cookbook repository. For more information, see Adding Apps.

OpsWorks Stacks passes the repository SSH key to each instance, and the built-in recipes then use the key to connect to the repository and download the code. The key is stored in the deploy attributes as `node[:deploy]['appshortname'][:scm][:ssh_key]`, and is accessible only to the root user.

Using Custom Domains

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

If you host a domain name with a third party, you can map that domain name to an app. The basic procedure is as follows:

1. Create a subdomain with your DNS registrar and map it to your load balancer's Elastic IP address or your app server's public IP address.
2. Update your app's configuration to point to the subdomain and redeploy the app.

> **Note**
>
> Make sure you forward your unqualified domain name (such as myapp1.example.com) to your qualified domain name (such as www.myapp1.example.com) so that both map to your app.

When you configure a domain for an app, it is listed as a server alias in the server's configuration file. If you are using a load balancer, the load balancer checks the domain name in the URL as requests come in and redirects the traffic based on the domain.

To map a subdomain to an IP address

1. If you are using a load balancer, on the **Instances** page, click the load balancer instance to open its details page and get the instance's **Elastic IP** address. Otherwise, get the public IP address from the application server instance's details page.
2. Follow the directions provided by your DNS registrar to create and map your subdomain to the IP address from Step 1.

> **Note**
>
> If the load balancer instance terminates at some point, you are assigned a new Elastic IP address. You need to update your DNS registrar settings to map to the new Elastic IP address.

OpsWorks Stacks simply adds the domain settings to the app's `deploy` attributes. You must implement a custom recipe to retrieve the information from the node object and configure the server appropriately. For more information, see Cookbooks and Recipes.

Running Multiple Applications on the Same Application Server

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

The information in this topic does not apply to Node.js apps.

If you have multiple applications of the same type, it is sometimes more cost-effective to run them on the same application server instances.

To run multiple applications on the same server

1. Add an app to the stack for each application.
2. Obtain a separate subdomain for each app and map the subdomains to the application server's or load balancer's IP address.
3. Edit each app's configuration to specify the appropriate subdomain.

For more information on how to perform these tasks, see Using Custom Domains.

Note

If your application servers are running multiple HTTP applications, you can use Elastic Load Balancing for load-balancing. For multiple HTTPS applications, you must either terminate the SSL connection at the load balancer or create a separate stack for each application. HTTPS requests are encrypted, which means that if you terminate the SSL connection

at the servers, the load balancer cannot check the domain name to determine which application should handle the request.

Using SSL

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

To use SSL with your application, you must first obtain a digital server certificate from a Certificate Authority (CA). For simplicity, this walkthrough creates a certificate and then self-signs it. Self-signed certificates are useful for learning and testing purposes, but you should always use a certificate signed by a CA for production stacks.

In this walkthrough, you'll do the following:

1. Install and configure OpenSSL.
2. Create a private key.
3. Create a certificate signing request.
4. Generate a self-signed certificate.
5. Edit the application with your certificate information.

Important

If your application uses SSL, we recommend that you disable SSLv3, if possible, in your application server layers to address the vulnerabilities described in CVE-2014-3566. If your stack includes a Ganglia layer, you should disable SSL v3 for that layer too. The details depend on the particular layer; for more information, see the following.

- Java App Server OpsWorks Stacks Layer

- Node.js App Server OpsWorks Stacks Layer
- PHP App Server OpsWorks Stacks Layer
- Rails App Server OpsWorks Stacks Layer
- Static Web Server OpsWorks Stacks Layer
- Ganglia Layer

Topics

- Step 1: Install and Configure OpenSSL
- Step 2: Create a Private Key
- Step 3: Create a Certificate Signing Request
- Step 4: Submit the CSR to Certificate Authority
- Step 5: Edit the App

Step 1: Install and Configure OpenSSL

Creating and uploading server certificates requires a tool that supports the SSL and TLS protocols. OpenSSL is an open-source tool that provides the basic cryptographic functions necessary to create an RSA token and sign it with your private key.

The following procedure assumes that your computer does not already have OpenSSL installed.

To install OpenSSL on Linux and Unix

1. Go to OpenSSL: Source, Tarballs.
2. Download the latest source.
3. Build the package.

To install OpenSSL on Windows

1. If the Microsoft Visual C++ 2008 Redistributable Package is not already installed on your system, download the package.
2. Run the installer and follow the instructions provided by the Microsoft Visual C++ 2008 Redistributable Setup Wizard to install the redistributable.

3. Go to OpenSSL: Binary Distributions, click the appropriate version of the OpenSSL binaries for your environment, and save the installer locally.
4. Run the installer and follow the instructions in the **OpenSSL Setup Wizard** to install the binaries.

Create an environment variable that points to the OpenSSL install point by opening a terminal or command window and using the following command lines.

- On Linux and Unix

```
export OpenSSL_HOME=path_to_your_OpenSSL_installation
```

- On Windows

```
set OpenSSL_HOME=path_to_your_OpenSSL_installation
```

Add the OpenSSL binaries' path to your computer's path variable by opening a terminal or command window and using the following command lines.

- On Linux and Unix

```
export PATH=$PATH:$OpenSSL_HOME/bin
```

- On Windows

```
set Path=OpenSSL_HOME\bin;%Path%
```

> **Note**
>
> Any changes you make to the environment variables by using these command lines are valid only for the current command-line session.

Step 2: Create a Private Key

You need a unique private key to create your Certificate Signing Request (CSR). Create the key by using the following command line:

```
openssl genrsa 2048 > privatekey.pem
```

Step 3: Create a Certificate Signing Request

A Certificate Signing Request (CSR) is a file sent to a Certificate Authority (CA) to apply for a digital server certificate. Create the CSR by using the following command line.

```
openssl req -new -key privatekey.pem -out csr.pem
```

The command's output will look similar to the following:

```
You are about to be asked to enter information that will be incorporated
 into your certificate request.
 What you are about to enter is what is called a Distinguished Name or a DN.
 There are quite a few fields but you can leave some blank
 For some fields there will be a default value,
 If you enter '.', the field will be left blank.
```

The following table can help you create your certificate request.

Certificate Request Data

Name	Description	Example
Country Name	The two-letter ISO abbreviation for your country.	US = United States
State or Province	The name of the state or province where your organization is located. This name cannot be abbreviated.	Washington
Locality Name	The name of the city where your organization is located.	Seattle
Organization Name	The full legal name of your organization. Do not abbreviate your organization name.	CorporationX

Name	Description	Example
Organizational Unit	(Optional) For additional organization information.	Marketing
Common Name	The fully qualified domain name for your CNAME. You will receive a certificate name check warning if this is not an exact match.	www.example.com
Email address	The server administrator's email address	someone@example.com

> **Note**
>
> The Common Name field is often misunderstood and is completed incorrectly. The common name is typically your host plus domain name. It will look like "www.example.com" or "example.com". You need to create a CSR using your correct common name.

Step 4: Submit the CSR to Certificate Authority

For production use, you would obtain a server certificate by submitting your CSR to a Certificate Authority (CA), which might require other credentials or proofs of identity. If your application is successful, the CA returns digitally signed identity certificate and possibly a certificate chain file. AWS does not recommend a specific CA. For a partial listing of available CAs, see Certificate Authority - Providers on Wikipedia.

You can also generate a self-signed certificate, which can be used for testing purposes only. For this example, use the following command line to generate a self-signed certificate.

```
openssl x509 -req -days 365 -in csr.pem -signkey privatekey.pem -out server.crt
```

The output will look similar to the following:

```
Loading 'screen' into random state - done
Signature ok
subject=/C=us/ST=washington/L=seattle/O=corporationx/OU=marketing/CN=example.com/
emailAddress=someone@example.com
Getting Private key
```

Step 5: Edit the App

After you generate your certificate and sign it, update your app to enable SSL and provide your certificate information. On the **Apps** page, choose an app to open the details page, and then click **Edit App**. To enable SSL support, set **Enable SSL** to **Yes**, which displays the following configuration options.

SSL Certificate

Paste the contents of the public key certificate (.crt) file into the box. The certificate should look something like the following:

```
-----BEGIN CERTIFICATE-----
MIICuTCCAiICCQCtqFKItVQJpzANBgkqhkiG9w0BAQUFADCBoDELMAkGA1UEBhMC
dXMxEzARBgNVBAgMCndhc2hpbmd0b24xEDAOBgNVBAcMB3NlYXR0bGUxDzANBgNV
BAoMBmFtYXpvbjEWMBQGA1UECwwNRGV2IGFuZCBUb29sczEdMBsGA1UEAwwUc3Rl
cGhhbmllYXBpZXJjZS5jb20xIjAgBgkqhkiG9w0BCQEWE3NhcGllcmNlQGFtYXpv
...
-----END CERTIFICATE-----
```

> **Note**
>
> If you are using Nginx and you have a certificate chain file, you should append the contents to the public key certificate file.

If you are updating an existing certificate, do the following:

- Choose **Update SSL certificate** to update the certificate.
- If the new certificate does not match the existing private key, choose **Update SSL certificate key**.
- If the new certificate does not match the existing certificate chain, choose **Update SSL certificates**.

SSL Certificate Key

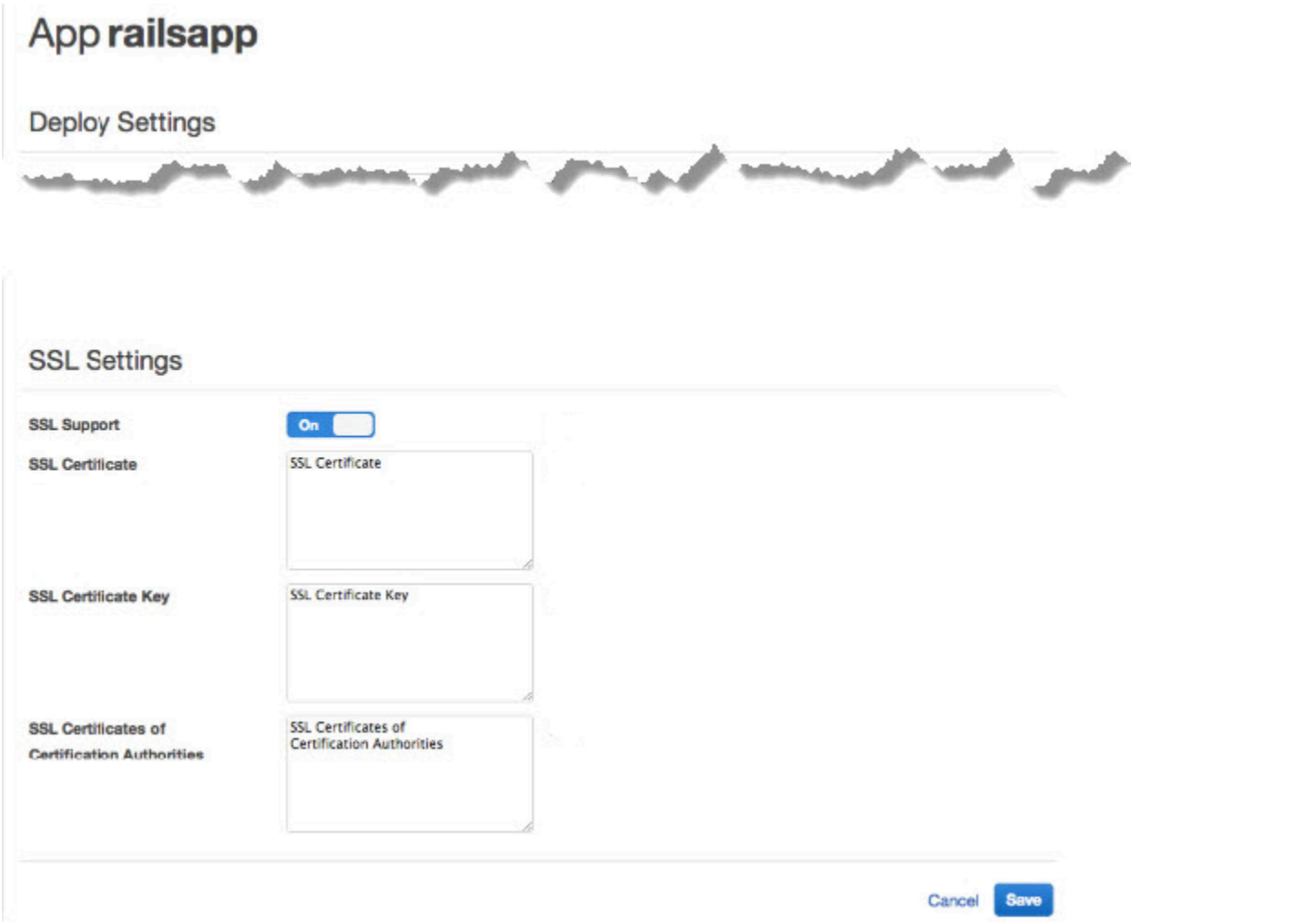
Paste the contents of the private key file (.pem file) into the box. It should look something like the following:

```
----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQC0CYklJY5r4vV2NHQYEpwtsLuMMBhylMrgBShKq+HHVLYQQCL6
+wGIiRq5qXqZlRXje3GM5Jvcm6q0R71MfRIl1FuzKyqDtneZaAIEYniZibHiUnmO
/UNqpFDosw/6hY3ONk0fSBlU4ivD0Gjpf6J80jL3DJ4R23Ed0sdL4pRT3QIDAQAB
AoGBAKmMfWrNRqYVtGKgnWB6Tji9QrKQLMXjmHeGg95mppdJELiXHhpMvrHtpIyK
...
-----END RSA PRIVATE KEY-----
```

SSL certificates of Certification Authorities

If you have a certificate chain file, paste the contents into the box.

> **Note**
>
> If you are using Nginx, you should leave this box empty. If you have a certificate chain file, append it to the public key certificate file in **SSL Certificate**.



After you click **Save**, redeploy the application to update your online instances.

For the built-in application server layers, OpsWorks Stacks automatically updates the server configuration. After deployment is finished, you can verify that your OpenSSL installation worked, as follows.

To verify an OpenSSL installation

1. Go to the **Instances** page.
2. Run the app by clicking the application server instance's IP address or, if you are using a load balancer, the load balancer's IP address.
3. Change the IP address prefix from **`http://`** to **`https://`** and refresh the browser to verify the page loads correctly with SSL.

Users who have configured apps to run in Mozilla Firefox sometimes get the following certificate error: `SEC_ERROR_UNKNOWN_ISSUER`. This error can be caused by certificate-replacement functionality in your organization's antivirus and antimalware programs, by some types of network traffic monitoring and filtering software, or by malware. For more information about how to troubleshoot this error, see How to troubleshoot security error codes on secure websites on the Mozilla Firefox Support website.

For all other layers, including custom layers, OpsWorks Stacks simply adds the SSL settings to the app's `deploy` attributes. You must implement a custom recipe to retrieve the information from the node object and configure the server appropriately.

Cookbooks and Recipes

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks uses Chef cookbooks to handle tasks such as installing and configuring packages and deploying apps. This section describes how to use cookbooks with OpsWorks Stacks. For more information, see Chef.

Note

OpsWorks Stacks currently supports Chef versions 12, 11.10.4, 11.4.4, and 0.9.15.5. However, Chef 0.9.15.5 is deprecated and we do not recommend that you use it for new stacks. For convenience, they are usually referred to by just their major and minor version numbers. Stacks running Chef 0.9 or 11.4 use Chef Solo and stacks running Chef 12 or 11.10 use Chef Client in local mode. For Linux stacks, you can use the Configuration Manager to specify which Chef version to use when you create a stack. Windows stacks must use Chef 12.2. For more information, including guidelines for migrating stacks to more recent Chef versions, see Chef Versions.

Topics

- Cookbook Repositories
- Chef Versions
- Ruby Versions
- Installing Custom Cookbooks
- Updating Custom Cookbooks
- Executing Recipes

Cookbook Repositories

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Your custom cookbooks must be stored in an online repository, either an archive such as a .zip file or a source control manager such as Git. A stack can have only one custom cookbook repository, but the repository can contain any number of cookbooks. When you install or update the cookbooks, OpsWorks Stacks installs the entire repository in a local cache on each of the stack's instances. When an instance needs, for example, to run one or more recipes, it uses the code from the local cache.

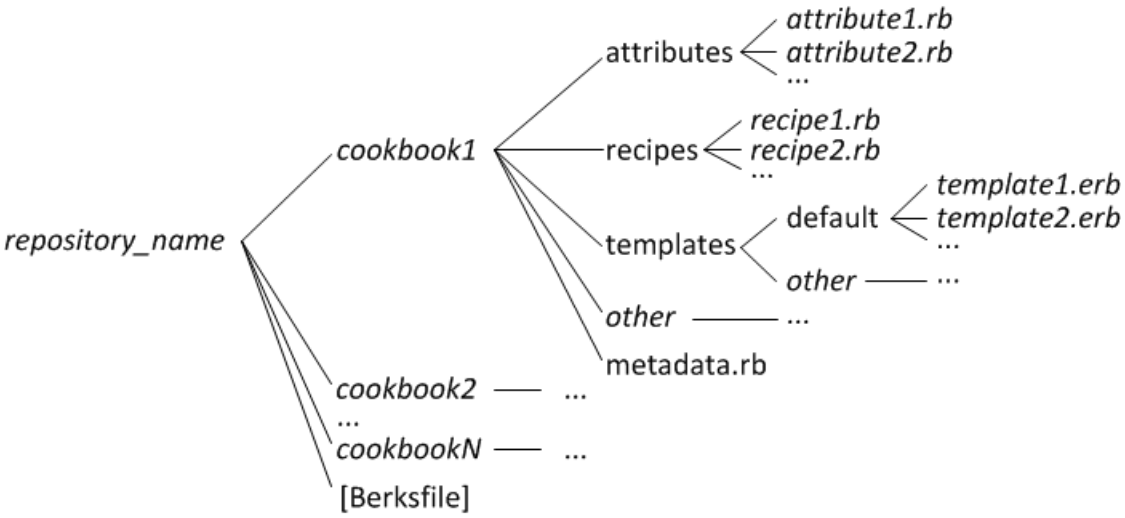
The following describes how to structure your cookbook repository, which depends on the type. The italicized text in the illustrations represents user-defined directory and file names, including the repository or archive name.

Source Control Manager

OpsWorks Stacks supports the following source control managers:

- Linux stacks – Git and Subversion
- Windows stacks – Git

The following shows the required directory and file structure:



- The cookbook directories must all be at the top-level.

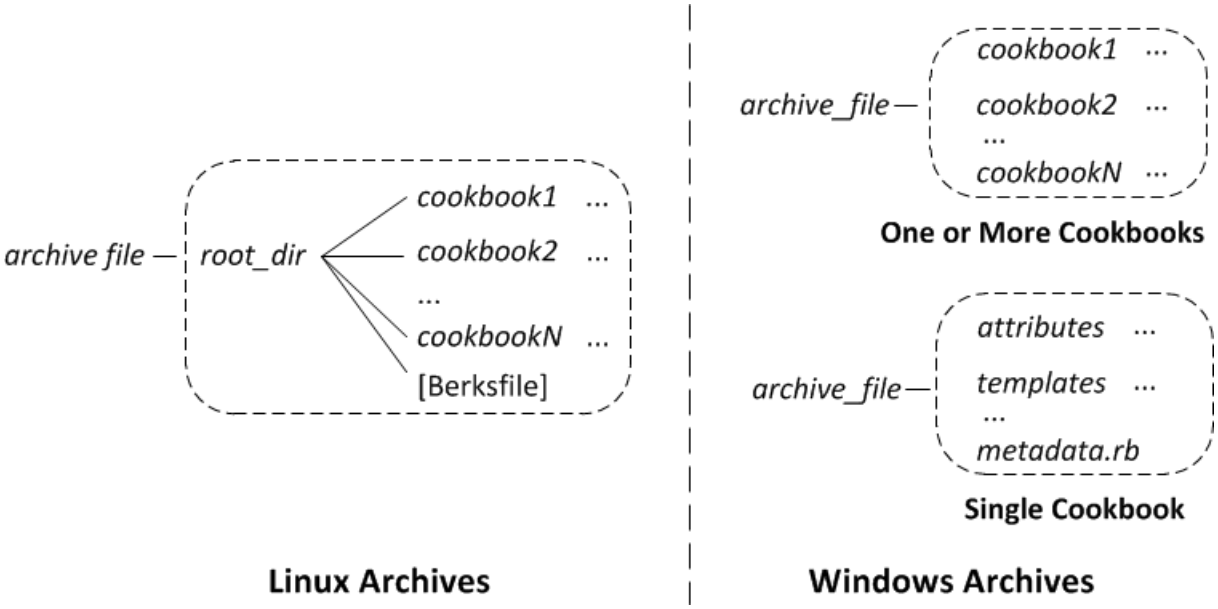
Archive

OpsWorks Stacks supports the following archives:

- Linux stacks – zip, gzip, bzip2, or tarball files, stored on Amazon S3 or a web site (HTTP archive).

  OpsWorks Stacks does not support uncompressed tarballs.
- Windows stacks – zip and tgz (gzip compressed tar) files, stored on Amazon S3.

The following shows the required directory and file structure, which depends on whether you are running a Linux or Windows stack. The cookbook structure is the same as for SCM repositories, so it is represented by an ellipsis (...).



- Linux stacks – The cookbook directories must be contained in a root directory.
- Windows stacks – The cookbooks must be at the archive's top level.

  If you have only one cookbook, you can optionally omit the cookbook directory and put the cookbook files at the top level. In that case, OpsWorks Stacks obtains the cookbook name from metadata.rb.

Each cookbook directory has least one and typically all of the following standard directories and files, which must use standard names:

- `attributes` – The cookbook's attributes files.
- `recipes` – The cookbook's recipe files.
- `templates` – The cookbook's template files.
- *`other`* – Optional user-defined directories that contain other file types, such as definitions or specs.
- `metadata.rb` – The cookbook's metadata.

  For Chef 11.10 and later, if your recipes depend on other cookbooks, you must include corresponding `depends` statements in your cookbook's `metadata.rb` file. For example, if your cookbook includes a recipe with a statement such as `include_recipe anothercookbook::somerecipe`, your cookbook's `metadata.rb` file must include the following line: `depends "anothercookbook"`. For more information, see About Cookbook Metadata.

Templates must be in a subdirectory of the `templates` directory, which contains at least one and optionally multiple subdirectories. Those subdirectories can optionally have subdirectories as well.

- Templates usually have a `default` subdirectory, which contains the template files that Chef uses by default.
- *other* represents optional subdirectories that can be used for operating system-specific templates.
- Chef automatically uses the template from the appropriate subdirectory, based on naming conventions that are described in File Specificity. For example, for the Amazon Linux and Ubuntu operating systems, you can put operating system-specific templates in subdirectories named `amazon` or `ubuntu`, respectively.

The details of how you handle custom cookbooks depend on your preferred repository type.

To use an archive

1. Implement your cookbooks by using the folder structure shown in the preceding section.
2. Create a compressed archive and upload it to an Amazon S3 bucket or a website.

   If you update your cookbooks, you must create and upload a new archive file. Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

To use an SCM

1. Set up a Git or Subversion repository using the structure shown earlier.
2. Optionally, use the repository's version control features to implement multiple branches or versions.

   If you update your cookbooks, you can do so in a new branch and just direct OpsWorks to use the new version. You can also specify particular tagged versions. For details, see Specifying a Custom Cookbook Repository.

Installing Custom Cookbooks describes how to have OpsWorks Stacks install your cookbook repository on the stack's instances.

> **Important**
>
> After you update existing cookbooks in the repository, you must run the `update_cookbooks` stack command to direct OpsWorks Stacks to update each online instance's local cache. For more information, see Run Stack Commands.

Chef Versions

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks supports multiple versions of Chef. You select the version when you create the stack. OpsWorks Stacks then installs that version of Chef on all of the stack's instances along with a set of built-in recipes that are compatible with that version. If you install any custom recipes, they must be compatible with the stack's Chef version.

OpsWorks Stacks currently supports Chef versions 12, 11.10, 11.4 and 0.9 for Linux stacks, and Chef 12.2 (currently Chef 12.22) for Windows stacks. For convenience, they are usually referred to by just their major and minor version numbers. For Linux stacks, you can use the Configuration Manager to specify which Chef version to use when you create a stack. Windows stacks must use Chef 12.2. For more information, including guidelines for migrating stacks to more recent Chef versions, see Chef Versions. For complete version information, see OpsWorks Stacks operating systems.

Chef 12.2

Chef 12.2 support was introduced in May 2015, and is used only by Windows stacks. The current version of Chef on Windows stacks is Chef 12.22. It runs with Ruby 2.3.6, and uses chef-client in local mode, which launches a local in-memory Chef server called chef-zero. The presence of this server enables recipes to use Chef search and data bags. The support has some limitations, which are described in Implementing Recipes: Chef 12.2, but you can run many community cookbooks without modification.

Chef 12

Chef 12 support was introduced in December 2015, and is used only by Linux stacks. It runs with Ruby 2.1.6 or 2.2.3 and uses chef-client in local mode, which enables recipes to use Chef search and data bags. For more information, see OpsWorks Stacks operating systems.

Chef 11.10

Chef 11.10 support was introduced in March 2014, and is used only by Linux stacks. It runs with Ruby 2.0.0 and uses chef-client in local mode, which enables recipes to use Chef search and data bags. The support has some limitations, which are described in Implementing Recipes: Chef 11.10, but you can run many community cookbooks without modification. You can also use Berkshelf to manage your cookbook dependencies. The supported Berkshelf versions

depend on the operating system. For more information, see OpsWorks Stacks operating systems. You cannot create CentOS stacks that use Chef 11.10.

Chef 11.4

Chef 11.4 support was introduced in July 2013, and is used only by Linux stacks. It runs with Ruby 1.8.7 and uses chef-solo, which does not support Chef search or data bags. You can often use community cookbooks that depend on those features with OpsWorks Stacks, but you must modify them as described in Migrating to a new Chef Version. You cannot create CentOS stacks that use Chef 11.4. Chef 11.4 stacks are not supported in regional endpoints outside the US East (N. Virginia) Region.

Chef 0.9

Chef 0.9 is used only by Linux stacks and is no longer supported. Note these details:

- You cannot use the console to create a new Chef 0.9 stack.

  You must use the CLI or API, or you must create a stack with a different Chef version and then edit the stack configuration.
- New OpsWorks Stacks features are not available for Chef 0.9 stacks.
- New operating system versions will provide only limited support for Chef 0.9 stacks.

  In particular, Amazon Linux 2014.09 and later versions do not support Chef 0.9 stacks with Rails App Server layers that depend on Ruby 1.8.7.
- New AWS regions, including Europe (Frankfurt), do not support Chef 0.9 stacks.

> **Note**
>
> We do not recommend using Chef 0.9 for new stacks. You should migrate any existing stacks to the latest Chef version as soon as possible.

If you want to use community cookbooks with OpsWorks Stacks, we recommend that you specify Chef 12 for new Linux stacks and migrate your existing Linux stacks to Chef 12. You can use the OpsWorks Stacks console, API, or CLI to migrate your existing stacks to a newer Chef version. For more information, see Migrating to a new Chef Version.

Topics

- Implementing Recipes for Chef 12.2 Stacks

- Implementing Recipes for Chef 12 Stacks
- Implementing Recipes for Chef 11.10 Stacks
- Implementing Recipes for Chef 11.4 Stacks
- Migrating an Existing Linux Stack to a new Chef Version

Implementing Recipes for Chef 12.2 Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Chef 12.2 (currently Chef 12.22) is available only on Windows stacks, which must run that Chef version.

- Recipes must use Windows-specific attributes and resources for some purposes.

  For more information, see Chef for Microsoft Windows.
- Chef runs use Ruby 2.3.6, so your recipes can use the new Ruby syntax.
- Recipes can use Chef search and data bags.

  Chef 12.2 stacks can use many community cookbooks without modification. For more information, see Using Chef Search and Using Data Bags.
- Most of the stack configuration and deployment attributes described in OpsWorks Stacks Data Bag Reference and Built-in Cookbook Attributes are available to Windows recipes.

  You can use Chef search to obtain these attribute values. For an example, see Obtaining Attribute Values with Chef Search. For a list of attributes, see OpsWorks Stacks Data Bag Reference.

Implementing Recipes for Chef 12 Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Chef 12 stacks provide the following advantages over Chef 11.10 stacks:

- Chef runs use Ruby 2.1.6, so your recipes can use the new Ruby syntax.
- Chef 12 stacks can use even more community cookbooks without modification. Without any built-in cookbooks in the way, there will no longer be any chance of name conflicts between built-in cookbooks and custom cookbooks.
- You are no longer limited to the Berkshelf versions that OpsWorks Stacks has provided pre-built packages for. Berkshelf is no longer installed on OpsWorks Stacks instances in Chef 12. Instead, you can use any Berkshelf version on your local workstation.
- There is now a clear separation between the built-in cookbooks that OpsWorks Stacks provides with Chef 12 (Elastic Load Balancing, Amazon RDS, and Amazon ECS) and custom cookbooks. This makes troubleshooting failed Chef runs easier.

Implementing Recipes for Chef 11.10 Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Chef 11.10 stacks provide the following advantages over Chef 11.4 stacks:

- Chef runs use Ruby 2.0.0, so your recipes can use the new Ruby syntax.
- Recipes can use Chef search and data bags.

  Chef 11.10 stacks can use many community cookbooks without modification.
- You can use Berkshelf to manage cookbooks.

  Berkshelf provides a much more flexible way to manage your custom cookbooks and to use community cookbooks in a stack.
- Cookbooks must declare dependencies in `metadata.rb`.

  If your cookbook depends on another cookbook, you must include that dependency in your cookbook's `metadata.rb` file. For example, if your cookbook includes a recipe with a statement such as `include_recipe anothercookbook::somerecipe`, your cookbook's `metadata.rb` file must include the following line: `depends "anothercookbook"`.
- OpsWorks Stacks installs a MySQL client on a stack's instances only if the stack includes a MySQL layer.
- OpsWorks Stacks installs a Ganglia client on a stack's instances only if the stack includes a Ganglia layer.
- If a deployment runs `bundle install` and the install fails, the deployment also fails.

> **Important**
>
> Do not reuse built-in cookbook names for custom or community cookbooks. Custom cookbooks that have the same name as built-in cookbooks might fail. For a complete list of built-in cookbooks that are available with Chef 11.10, 11.4, and 0.9 stacks, see the opsworks-cookbooks repository on GitHub.
> Cookbooks with non-ASCII characters that run successfully on Chef 0.9 and 11.4 stacks might fail on a Chef 11.10 stack. The reason is that Chef 11.10 stacks use Ruby 2.0.0 for Chef runs, which is much stricter about encoding than Ruby 1.8.7. To ensure that such cookbooks run successfully on Chef 11.10 stacks, each file that uses non-ASCII characters should have a comment at the top that provides a hint about the encoding. For example, for UTF-8 encoding, the comment would be `# encoding: UTF-8`. For more information on Ruby 2.0.0 encoding, see Encoding.

Topics

- Cookbook Installation and Precedence
- Using Chef Search
- Using Data Bags
- Using Berkshelf

Cookbook Installation and Precedence

The procedure for installing OpsWorks Stacks cookbooks works somewhat differently for Chef 11.10 stacks than for earlier Chef versions. For Chef 11.10 stacks, after OpsWorks Stacks installs the built-in, custom, and Berkshelf cookbooks, it merges them to a common directory in the following order:

1. Built-in cookbooks.
2. Berkshelf cookbooks, if any.
3. Custom cookbooks, if any.

When OpsWorks Stacks performs this merge, it copies the entire contents of the directories, including recipes. If there are any duplicates, the following rules apply:

- The contents of Berkshelf cookbooks take precedence over the built-in cookbooks.
- The contents of custom cookbooks take precedence over the Berkshelf cookbooks.

To illustrate how this process works, consider the following scenario, where all three cookbook directories include a cookbook named `mycookbook`:

- Built-in cookbooks – `mycookbook` includes an attributes file named `someattributes.rb`, a template file named `sometemplate.erb`, and a recipe named `somerecipe.rb`.
- Berkshelf cookbooks – `mycookbook` includes `sometemplate.erb` and `somerecipe.rb`.
- Custom cookbooks – `mycookbook` includes `somerecipe.rb`.

The merged cookbook contains the following:

- `someattributes.rb` from the built-in cookbook.
- `sometemplate.erb` from the Berkshelf cookbook.
- `somerecipe.rb` from the custom cookbook.

Important

You should not customize your Chef 11.10 stack by copying an entire built-in cookbook to your repository and then modifying parts of the cookbook. Doing so overrides the entire built-in cookbook, including recipes. If OpsWorks Stacks updates that cookbook, your stack will not get the benefit of those updates unless you manually update your private copy. For more information on how to customize stacks, see Customizing OpsWorks Stacks.

Using Chef Search

You can use the Chef `search` Method in your recipes to query for stack data. You use the same syntax as you would for Chef server, but OpsWorks Stacks obtains the data from the local node object instead of querying a Chef server. This data includes:

- The instance's stack configuration and deployment attributes.
- The attributes from the instance's built-in and custom cookbooks' attributes files.
- System data collected by Ohai.

The stack configuration and deployment attributes contain most of the information that recipes typically obtain through search, including data such as host names and IP addresses for each online instance in the stack. OpsWorks Stacks updates these attributes for each lifecycle event, which ensures that they accurately reflect the current stack state. This means that you can often use search-dependent community recipes in your stack without modification. The search method still returns the appropriate data; it's just coming from the stack configuration and deployment attributes instead of a server.

The primary limitation of OpsWorks Stacks search is that handles only the data in the local node object, the stack configuration and deployment attributes in particular. For that reason, the following types of data might not be available through search:

- Locally defined attributes on other instances.

  If a recipe defines an attribute locally, that information is not reported back to the OpsWorks Stacks service, so you cannot access that data from other instances by using search.
- Custom `deploy` attributes.

You can specify custom JSON when you deploy an app and the corresponding attributes are installed on the stack's instances for that deployment. However, if you deploy only to selected instances, the attributes are installed on only those instances. Queries for those custom JSON attributes will fail on all other instances. In addition, the custom attributes are included in the stack configuration and deployment JSON only for that particular deployment. They are accessible only until the next lifecycle event installs a new set of stack configuration and deployment attributes. Note that if you specify custom JSON for the stack, the attributes are installed on every instance for every lifecycle event and are always accessible through search.

- Ohai data from other instances.

  Chef's Ohai tool obtains a variety of system data on an instance and adds it to the node object. This data is stored locally and not reported back to the OpsWorks Stacks service, so search can't access Ohai data from other instances. However, some of this data might be included in the stack configuration and deployment attributes.

- Offline instances.

  The stack configuration and deployment attributes contain data only for online instances.

The following recipe excerpt shows how to get the private IP address of a PHP layer's instance by using search.

```
appserver = search(:node, "role:php-app").first
Chef::Log.info("The private IP is '#{appserver[:private_ip]}'")
```

Note

When OpsWorks Stacks adds the stack configuration and deployment attributes to the node object, it actually creates two sets of layer attributes, each with the same data. One set is in the `layers` namespace, which is how OpsWorks Stacks stores the data. The other set is in the `role` namespace, which is how Chef server stores the equivalent data. The purpose of the `role` namespace is to allow search code that was implemented for Chef server to run on an OpsWorks Stacks instance. If you are writing code specifically for OpsWorks Stacks, you could use either `layers:php-app` or `role:php-app` in the preceding example and `search` would return the same result.

Using Data Bags

You can use the Chef `data_bag_item` method in your recipes to query for information in a data bag. You use the same syntax as you would for Chef server, but OpsWorks Stacks obtains the data from the instance's stack configuration and deployment attributes. However, OpsWorks Stacks does not currently support Chef environments, so `node.chef_environment` always returns `_default`.

You create a data bag by using custom JSON to add one or more attributes to the `[:opsworks][:data_bags]` attribute. The following example shows the general format for creating a data bag in custom JSON.

> **Note**
>
> You cannot create a data bag by adding it to your cookbook repository. You must use custom JSON.

```
{
  "opsworks": {
    "data_bags": {
      "bag_name1": {
        "item_name1: {
          "key1" : “value1”,
          "key2" : “value2”,
          ...
        }
      },
      "bag_name2": {
        "item_name1": {
          "key1" : “value1”,
          "key2" : “value2”,
          ...
        }
      },
      ...
    }
  }
}
```

You typically specify custom JSON for the stack, which installs the custom attributes on every instance for each subsequent lifecycle event. You can also specify custom JSON when you deploy an app, but those attributes are installed only for that deployment, and might be installed to only a selected set of instances. For more information, see Deploying Apps.

The following custom JSON example creates data bag named `myapp`. It has one item, `mysql`, with two key-value pairs.

```
{ "opsworks": {
    "data_bags": {
      "myapp": {
        "mysql": {
          "username": "default-user",
          "password": "default-pass"
        }
      }
    }
  }
}
```

To use the data in your recipe, you can call `data_bag_item` and pass it the data bag and value names, as shown in the following excerpt.

```
mything = data_bag_item("myapp", "mysql")
Chef::Log.info("The username is '#{mything['username']}' ")
```

To modify the data in the data bag, just modify the custom JSON, and it will be installed on the stack's instances for the next lifecycle event.

Using Berkshelf

With Chef 0.9 and Chef 11.4 stacks, you can install only one custom cookbook repository. With Chef 11.10 stacks, you can use Berkshelf to manage your cookbooks and their dependencies, which allows you to install cookbooks from multiple repositories. (For more information, see Packaging Cookbook Dependencies Locally.) In particular, with Berkshelf, you can install OpsWorks Stacks-compatible community cookbooks directly from their repositories instead of having to copy them to your custom cookbook repository. The supported Berkshelf versions depend on the operating system. For more information, see OpsWorks Stacks operating systems.

To use Berkshelf, you must explicitly enable it, as described in Installing Custom Cookbooks. Then, include a `Berksfile` file in your cookbook repository's root directory that specifies which cookbooks to install.

To specify an external cookbook source in a Berksfile, include a source attribute at the top of the file that specifies the default repository URL. Berkshelf will look for the cookbooks in the source URLs unless you explicitly specify a repository. Then include a line for each cookbook that you want to install in the following format:

```
cookbook 'cookbook_name', ['>= cookbook_version'], [cookbook_options]
```

The fields following `cookbook` specify the particular cookbook.

- *`cookbook_name`* – (Required) Specifies the cookbook's name.

  If you don't include any other fields, Berkshelf installs the cookbook from the specified source URLs.
- *`cookbook_version`* – (Optional) Specifies the cookbook version or versions.

  You can use a prefix such as = or >= to specify a particular version or a range of acceptable versions. If you don't specify a version, Berkshelf installs the latest one.
- *`cookbook_options`* – (Optional) The final field is a hash containing one or more key-value pairs that specify options such as the repository location.

  For example, you can include a `git` key to designate a particular Git repository and a `tag` key to designate a particular repository branch. Specifying the repository branch is usually the best way to ensure that you install your preferred cookbook.

> **Important**
>
> Do not declare cookbooks by including a `metadata` line in your Berksfile and declaring the cookbook dependencies in `metadata.rb`. For this to work correctly, both files must be in the same directory. With OpsWorks Stacks, the Berksfile must be in the repository's root directory, but `metadata.rb` files must be in their respective cookbook directories. You should instead explicitly declare external cookbooks in the Berksfile.

The following is an example of a Berksfile that shows different ways to specify cookbooks For more information on how to create a Berksfile, see Berkshelf.

```
source "https://supermarket.chef.io"

cookbook 'apt'
cookbook 'bluepill', '>= 2.3.1'
cookbook 'ark', git: 'git://github.com/opscode-cookbooks/ark.git'
cookbook 'build-essential', '>= 1.4.2', git: 'git://github.com/opscode-cookbooks/build-essential.git', tag: 'v1.4.2'
```

This file installs the following cookbooks:

- The most recent version of `apt` from the community cookbooks repository.
- The most recent version `bluepill` from the community cookbooks, as long as it is version 2.3.1 or later.
- The most recent version of `ark` from a specified repository.

  The URL for this example is for a public community cookbook repository on GitHub, but you can install cookbooks from other repositories, including private repositories. For more information, see Berkshelf.
- The `build-essential` cookbook from the v1.4.2 branch of the specified repository.

A custom cookbook repository can contain custom cookbooks in addition to a Berksfile. In that case, OpsWorks Stacks installs both sets of cookbooks, which means that an instance can have as many as three cookbook repositories.

- The built-in cookbooks are installed to `/opt/aws/opsworks/current/cookbooks`.
- If your custom cookbook repository contains cookbooks, they are installed to `/opt/aws/opsworks/current/site-cookbooks`.
- If you have enabled Berkshelf and your custom cookbook repository contains a Berksfile, the specified cookbooks are installed to `/opt/aws/opsworks/current/berkshelf-cookbooks`.

The built-in cookbooks and your custom cookbooks are installed on each instance during setup and are not subsequently updated unless you manually run the **Update Custom Cookbooks** stack

command. OpsWorks Stacks runs `berks install` for every Chef run, so your Berkshelf cookbooks are updated for each lifecycle event, according to the following rules:

- If you have a new cookbook version in the repository, this operation updates the cookbook from the repository.
- Otherwise, this operation updates the Berkshelf cookbooks from a local cache.

> **Note**
>
> The operation overwrites the Berkshelf cookbooks, so if you have modified the local copies of any cookbooks, the changes will be overwritten. For more information, see Berkshelf

You can also update your Berkshelf cookbooks by running the **Update Custom Cookbooks** stack command, which updates both the Berkshelf cookbooks and your custom cookbooks.

Implementing Recipes for Chef 11.4 Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Important**
>
> Do not reuse built-in cookbook names for custom or community cookbooks. Custom cookbooks that have the same name as built-in cookbooks might fail. For a complete list of built-in cookbooks that are available with Chef 11.10, 11.4, and 0.9 stacks, see the opsworks-cookbooks repository on GitHub.

The primary limitation of Chef 11.4 stacks is that recipes cannot use Chef search or data bags. However, OpsWorks Stacks installs stack configuration and deployment attributes on each instance that contain much of the information that you would obtain with search, including the following:

- User-defined data from the console such as host or app names.
- Stack configuration data generated by the OpsWorks Stacks service, such as the stack's layers, apps, and instances, and details about each instance such as the IP address.
- Custom JSON attributes that contain data provided by the user and can serve much the same purpose as data bags.

OpsWorks Stacks installs a current version of the stack configuration and deployment attributes on each instance for each lifecycle event, prior to starting the event's Chef run. The data is available to recipes through the standard `node[:attribute][:child_attribute][...]` syntax. For example, the stack configuration and deployment attributes includes the stack name, `node[:opsworks][:stack][:name]`.

The following excerpt from one of the built-in recipes obtains the stack name and uses it to create a configuration file.

```
template '/etc/ganglia/gmetad.conf' do
  source 'gmetad.conf.erb'
  mode '0644'
  variables :stack_name => node[:opsworks][:stack][:name]
  notifies :restart, "service[gmetad]"
end
```

Many of the stack configuration and deployment attribute values contain multiple attributes. You must iterate over these attributes to obtain the information you need. The example below shows an excerpt from the stack configuration and deployment attributes, which are represented as JSON object for convenience. It contains a top-level attribute, `deploy`, which contains an attribute for each of the stack's apps, named with the app's short name.

```
{
  ...
  "deploy": {
    "app1_shortname": {
      "document_root": "app1_root",
```

```
      "deploy_to": "deploy_directory",
      "application_type": "php",
      ...
    },
    "app2_shortname": {
      "document_root": "app2_root",
      ...
    }
  },
  ...
}
```

Each app attribute contains a set of attributes that characterize the app. For example, the `deploy_to` attribute represents the app's deploy directory. The following excerpt sets the user, group, and path for each app's deploy directory.

```
node[:deploy].each do |application, deploy|
  opsworks_deploy_dir do
    user deploy[:user]
    group deploy[:group]
    path deploy[:deploy_to]
  end
  ...
end
```

For more information on the stack configuration and deployment attributes, see Customizing OpsWorks Stacks. For more information on deploy directories, see Deploy Recipes.

Chef 11.4 stacks do not support data bags, but you can add arbitrary data to the stack configuration and deployment attributes by specifying custom JSON. Your recipes can then access the data by using standard Chef node syntax. For more information, see Using Custom JSON.

If you need the functionality of an encrypted data bag, one option is to store sensitive attributes in a secure location such as a private Amazon S3 bucket. Your recipes can then use the AWS Ruby SDK—which is installed on all OpsWorks Stacks instances—to download the data from the bucket.

Note

Each OpsWorks Stacks instance has an instance profile. The associated IAM role specifies which AWS resources can be accessed by applications that are running on the instance. For

your recipes to access an Amazon S3 bucket, the role's policy must include a statement similar to the following, which grants permission to retrieve files from a specified bucket.

```
"Action": ["s3:GetObject"],
"Effect": "Allow",
"Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",
```

For more information on instance profiles, see Specifying Permissions for Apps Running on EC2 instances.

Migrating an Existing Linux Stack to a new Chef Version

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can use OpsWorks Stacks console, API, or CLI to migrate your Linux stacks to a newer Chef version. However, your recipes might require modification to be compatible with the newer version. When preparing to migrate a stack, consider the following.

- You cannot change OpsWorksStacks stack versions from Chef 11 to Chef 12 by editing or cloning the stack. A Chef major version upgrade cannot be performed using the procedure in this section. For more information on the Chef 11.10 to Chef 12 transition, see Implementing Recipes: Chef 12.
- The transition from one Chef version to another involves a number of changes, some of them breaking changes.

  For more information on the Chef 0.9 to Chef 11.4 transition, see Migrating to a new Chef Version. For more information on the Chef 11.4 to Chef 11.10 transition, see Implementing Recipes: Chef 11.10. For more information on the Chef 11.10 to Chef 12 transition, see Implementing Recipes: Chef 12.

- Chef runs use a different Ruby version on Chef 0.9 and Chef 11.4 stacks (Ruby 1.8.7), Chef 11.10 stacks (Ruby 2.0.0), and Chef 12 stacks (Ruby 2.1.6).

  For more information, see Ruby Versions.

- Chef 11.10 stacks handle cookbook installation differently from Chef 0.9 or Chef 11.4 stacks.

  This difference could cause problems when migrating stacks that use custom cookbooks to Chef 11.10. For more information, see Cookbook Installation and Precedence.

The following are recommended guidelines for migrating a Chef stack to a newer Chef version:

To migrate a stack to a newer Chef version

1. Clone your production stack. On the **Clone Stack** page, click **Advanced>>** to display the **Configuration Management** section, and change **Chef version** to the next higher version.

   > **Note**
   >
   > If you are starting with a Chef 0.9 stack, you cannot upgrade directly to Chef 11.10; you must first upgrade to Chef 11.4. If you want to migrate your stack to Chef 11.10 before testing your recipes, wait 20 minutes for the update to be executed, and then upgrade the stack from 11.4 to 11.10.

2. Add instances to the layers and test the cloned stack's applications and cookbooks on a testing or staging system. For more information, see All about Chef ....
3. When the test results are satisfactory, do one of the following:

   - If this is your desired Chef version, you can use the cloned stack as your production stack, or reset the Chef version on your production stack.
   - If you are migrating a Chef 0.9 stack to Chef 11.10 in two stages, repeat the process to migrate the stack from Chef 11.4 to Chef 11.10.

> **Note**
>
> When you are testing recipes, you can use SSH to connect to the instance and then use the Instance Agent CLI run_command command to run the recipes associated with the various lifecycle events. The agent CLI is especially useful for testing Setup recipes because you

can use it even Setup fails and the instance does not reach the online state. You can also use the Setup stack command to rerun Setup recipes, but that command is only available if Setup succeeded and the instance is online.

It is possible to update a running stack to a new Chef version.

To update a running stack to a new Chef version

1. Edit the stack to change the **Chef version** stack setting.
2. Save the new settings and wait for OpsWorks Stacks to update the instances, which typically takes 15 - 20 minutes.

Important

OpsWorks Stacks does not synchronize the Chef version update with lifecycle events. If you want to update the Chef version on a production stack, you must take care to ensure that the update is complete before the next lifecycle event occurs . If an event occurs—typically a Deploy or Configure event—the instance agent updates your custom cookbooks and runs the event's assigned recipes, whether the version update is complete or not. There is no direct way to determine when the version update is complete, but deployment logs include the Chef version.

Ruby Versions

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

All instances in a Linux stack have Ruby installed. OpsWorks Stacks installs a Ruby package on each instance, which it uses to run Chef recipes and the instance agent. OpsWorks Stacks determines

the Ruby version based on which Chef version the stack is running. Do not attempt to modify this version; doing so might disable the instance agent.

OpsWorks Stacks does not install an application Ruby executable on Windows stacks. The Chef 12.2 client comes with Ruby 2.0.0 p451, but the Ruby executable is not added to the instances' PATH environment variable. If you want to use this executable to run Ruby code, it is located at `\opscode\chef\embedded\bin\ruby.exe` on your Windows drive.

The following table summarizes OpsWorks Stacks Ruby versions. The available application Ruby versions also depend on the instance's operating system. For more information, including the available patch versions, see OpsWorks Stacks operating systems.

Chef Version	Chef Ruby Version	Available Application Ruby Versions
0.9 (c)	1.8.7	1.8.7(a), 1.9.3(e), 2.0.0
11.4 (c)	1.8.7	1.8.7(a), 1.9.3(e), 2.0.0, 2.1, 2.2.0, 2.3
11.10	2.0.0-p481	1.9.3(c, e), 2.0.0, 2.1, 2.2.0, 2.3, 2.6.1
12 (b)	2.1.6, 2.2.3	None
12.22 (d)	2.3.6	None

(a) Not available with Amazon Linux 2014.09 and later, Red Hat Enterprise Linux (RHEL), or Ubuntu 14.04 LTS.

(b) Available only on Linux stacks.

(c) Not available with RHEL.

(d) Available only on Windows stacks. Major version is 12.2. Current minor version is 12.22.

(e) Deprecation is complete; support has ended.

The install locations depend on the Chef version:

- Applications use the `/usr/local/bin/ruby` executable for all Chef Versions.
- For Chef 0.9 and 11.4, the instance agent and Chef recipes use the `/usr/bin/ruby` executable.

- For Chef 11.10, the instance agent and Chef recipes use the `/opt/aws/opsworks/local/bin/ruby` executable.

Installing Custom Cookbooks

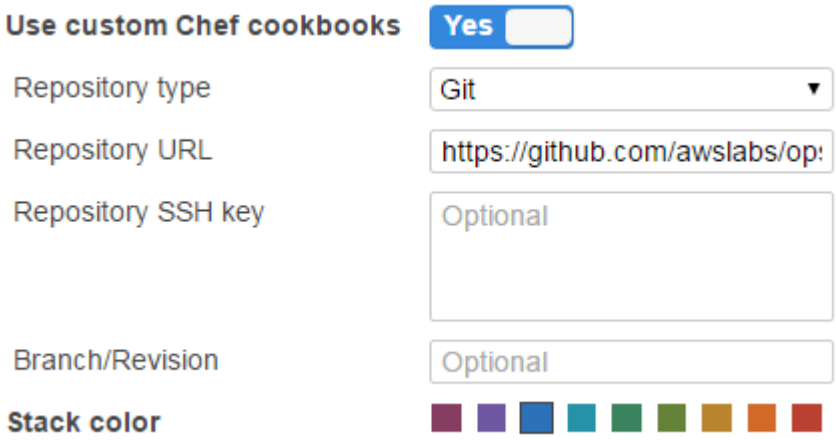
> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

To have a stack install and use custom cookbooks, you must configure the stack to enable custom cookbooks, if it is not already configured. You must then provide the repository URL and any related information such as a password.

> **Important**
>
> After you have configured the stack to support custom cookbooks, OpsWorks Stacks automatically installs your cookbooks on all new instances at startup. However, you must explicitly direct OpsWorks Stacks to install new or updated cookbooks on any existing instances by running the **Update Custom Cookbooks** stack command. For more information, see Updating Custom Cookbooks. Before you enable **Use custom Chef cookbooks** on your stack, be sure that custom and community cookbooks that you run support the version of Chef that your stack uses.

To configure a stack for custom cookbooks

1. On your stack's page, click **Stack Settings** to display its **Settings** page., Click **Edit** to edit the settings.
2. Toggle **Use custom Chef cookbooks** to **Yes**.



3. Configure your custom cookbooks.

When you are finished, click **Save** to save the updated stack.

Specifying a Custom Cookbook Repository

Linux stacks can install custom cookbooks from any of the following repository types:

- HTTP or Amazon S3 archives.

  They can be either public or private, but Amazon S3 is typically the preferred option for a private archive.
- Git and Subversion repositories provide source control and the ability to have multiple versions.

Windows stacks can install custom cookbooks from Amazon S3 archives and Git repositories.

All repository types have the following required fields.

- **Repository type**–The repository type
- **Repository URL**–The repository URL

OpsWorks Stacks supports publicly hosted Git repository sites such as GitHub or Bitbucket as well as privately hosted Git servers. For Git repositories, you must use one of the following URL formats, depending on whether the repository is public or private. Follow the same URL guidelines for Git submodules.

For a public Git repository, use the HTTPS or Git read-only protocols:

- Git read-only – `git://github.com/amazonwebservices/opsworks-example-cookbooks.git`.
- HTTPS – `https://github.com/amazonwebservices/opsworks-example-cookbooks.git`.

For a private Git repository, you must use the SSH read/write format, as shown in the following examples:

- Github repositories – `git@github.com:`*`project`*`/`*`repository`*.
- Repositories on a Git server – *`user`*`@`*`server`*`:`*`project`*`/`*`repository`*

The remaining settings vary with the repository type and are described in the following sections.

HTTP Archive

Selecting **Http Archive** for **Repository type** displays two additional settings, which you must complete if the archive is password protected.

- **User name**–Your user name
- **Password**–Your password

Amazon S3 Archive

Selecting **S3 Archive** for **Repository type** displays the following additional, optional settings. OpsWorks Stacks can access your repository by using Amazon EC2 roles (host operating system manager authentication), whether you use the OpsWorks Stacks API or console.

- **Access key ID** –An AWS access key ID, such as AKIAIOSFODNN7EXAMPLE.
- **Secret access key** – The corresponding AWS secret access key, such as wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY.

Git Repository

Selecting **Git** under **Source Control** displays the following additional optional settings:

Repository SSH key

You must specify a deploy SSH key to access private Git repositories. For Git submodules, the specified key must have access to those submodules. For more information, see Using Git Repository SSH Keys.

> **Important**
>
> The deploy SSH key cannot require a password; OpsWorks Stacks has no way to pass it through.

Branch/Revision

If the repository has multiple branches, OpsWorks Stacks downloads the master branch by default. To specify a particular branch, enter the branch name, SHA1 hash, or tag name. To specify a particular commit, enter the full 40-hexdigit commit ID.

Subversion Repository

Selecting **Subversion** under **Source Control** displays the following additional settings:

- **User name**–Your user name, for private repositories.
- **Password**–Your password, for private repositories.
- **Revision**–[Optional] The revision name, if you have multiple revisions.

To specify a branch or tag, you must modify the repository URL, for example: **`http://repository_domain/repos/myapp/branches/my-apps-branch`** or **`http://repository_domain_name/repos/calc/myapp/my-apps-tag`**.

Updating Custom Cookbooks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

When you provide OpsWorks Stacks with custom cookbooks, the built-in Setup recipes create a local cache on each newly-started instance, and download the cookbooks to the cache. OpsWorks Stacks then runs recipes from the cache, not the repository. If you modify the custom cookbooks in the repository, you must ensure that the updated cookbooks are installed on your instances' local caches. OpsWorks Stacks automatically deploys the latest cookbooks to new instances when they are started. For existing instances, however, the situation is different:

- You must manually deploy updated custom cookbooks to online instances.
- You do not have to deploy updated custom cookbooks to offline instance store-backed instances, including load-based and time-based instances.

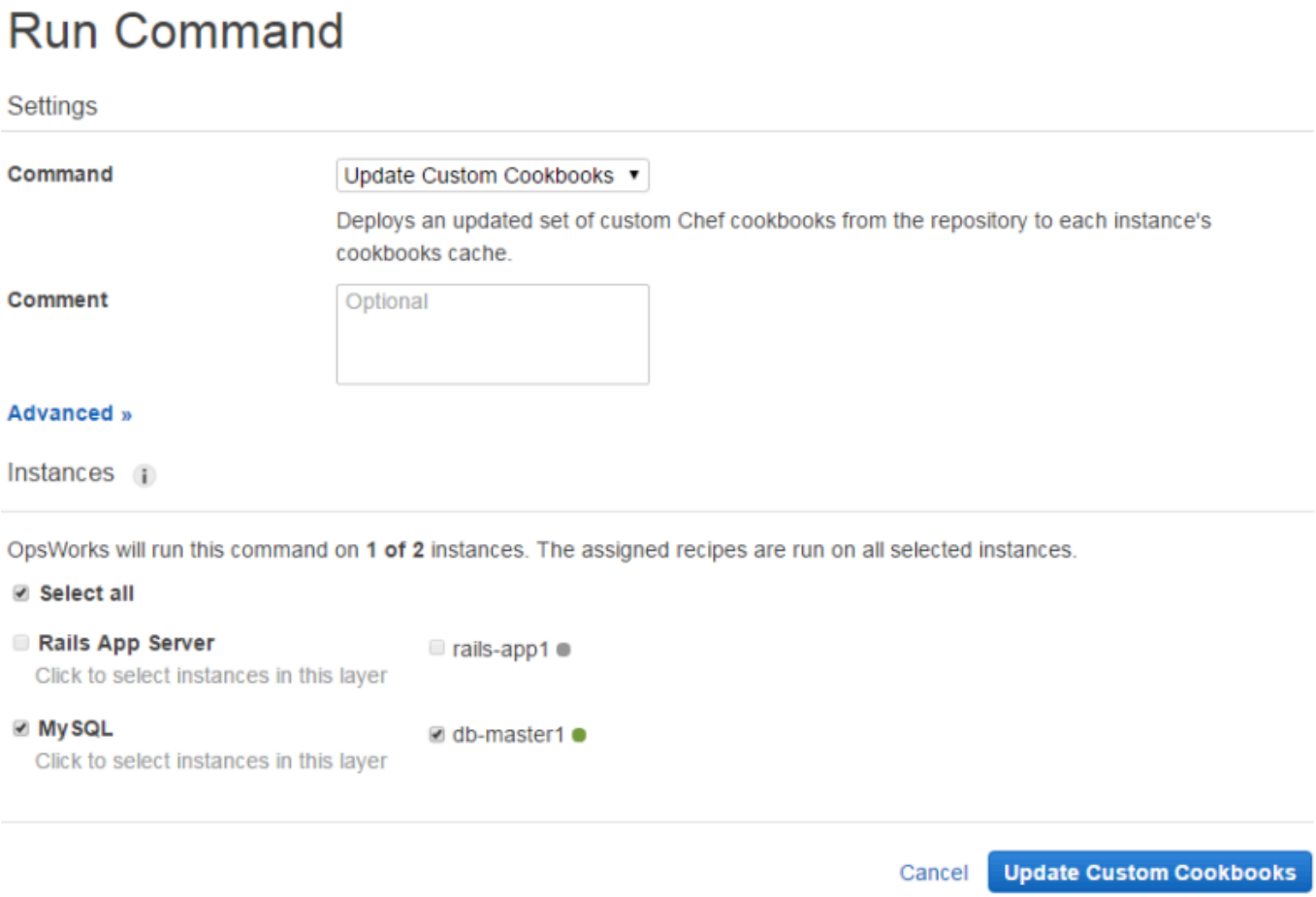
  OpsWorks Stacks automatically deploys the current cookbooks when the instances are restarted.
- You must start offline EBS-backed 24/7 instances that are not load-based or time-based.
- You cannot start offline EBS-backed load-based and time-based instances, so the simplest approach is to delete the offline instances and add new instances to replace them.

  Because they are now new instances, OpsWorks Stacks automatically deploys the current custom cookbooks when the instances are started.

To manually update custom cookbooks

1. Update your repository with the modified cookbooks. OpsWorks Stacks uses the cache URL that you provided when you originally installed the cookbooks, so the cookbook root file name, repository location, and access rights should not change.

   - For Amazon S3 or HTTP repositories, replace the original .zip file with a new .zip file that has the same name.
   - For Git or Subversion repositories, edit your stack settings to change the **Branch/Revision** field to the new version.
2. On the stack's page, click **Run Command** and select the **Update Custom Cookbooks** command.



3. Add a comment if desired.
4. Optionally, specify a custom JSON object for the command to add custom attributes to the stack configuration and deployment attributes that OpsWorks Stacks installs on the instances. For more information, see Using Custom JSON and Overriding Attributes.
5. By default, OpsWorks Stacks updates the cookbooks on every instance. To specify which instances to update, select the appropriate instances from the list at the end of the page. To select every instance in a layer, select the appropriate layer checkbox in the left column.
6. Click **Update Custom Cookbooks** to install the updated cookbooks. OpsWorks Stacks deletes the cached custom cookbooks on the specified instances and installs the new cookbooks from the repository.

> **Note**
>
> This procedure is required only for existing instances, which have old versions of the cookbooks in their caches. If you subsequently add instances to a layer, OpsWorks Stacks

deploys the cookbooks that are currently in the repository so they automatically get the latest version.

Executing Recipes

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can run recipes in two ways:

- Automatically, by assigning recipes to the appropriate layer's lifecycle event.
- Manually, by running the Execute Recipes stack command or by using the agent CLI.

Topics

- OpsWorks Stacks Lifecycle Events
- Automatically Running Recipes
- Manually Running Recipes

OpsWorks Stacks Lifecycle Events

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Each layer has a set of five lifecycle events, each of which has an associated set of recipes that are specific to the layer. When an event occurs on a layer's instance, OpsWorks Stacks automatically runs the appropriate set of recipes. To provide a custom response to these events, implement custom recipes and assign them to the appropriate events for each layer. OpsWorks Stacks runs those recipes after the event's built-in recipes.

Setup

This event occurs after a started instance has finished booting. You can also manually trigger the Setup event by using the Setup stack command. OpsWorks Stacks runs recipes that set the instance up according to its layer. For example, if the instance is a member of the Rails App Server layer, the Setup recipes install Apache, Ruby Enterprise Edition, Passenger and Ruby on Rails.

> **Note**
>
> A **Setup** event takes an instance out of service. Because an instance is not in the **Online** state when the **Setup** lifecycle event runs, instances on which you run **Setup** events are removed from a load balancer.

Configure

This event occurs on all of the stack's instances when one of the following occurs:

- An instance enters or leaves the online state.
- You associate an Elastic IP address with an instance or disassociate one from an instance.
- You attach an Elastic Load Balancing load balancer to a layer, or detach one from a layer.

For example, suppose that your stack has instances A, B, and C, and you start a new instance, D. After D has finished running its setup recipes, OpsWorks Stacks triggers the Configure event on A, B, C, and D. If you subsequently stop A, OpsWorks Stacks triggers the Configure event on B, C, and D. OpsWorks Stacks responds to the Configure event by running each layer's Configure recipes, which update the instances' configuration to reflect the current set of online instances. The Configure event is therefore a good time to regenerate configuration files. For example, the HAProxy Configure recipes reconfigure the load balancer to accommodate any changes in the set of online application server instances.

You can also manually trigger the Configure event by using the Configure stack command.

Deploy

This event occurs when you run a **Deploy** command, typically to deploy an application to a set of application server instances. The instances run recipes that deploy the application and any related files from its repository to the layer's instances. For example, for a Rails Application Server instances, the Deploy recipes check out a specified Ruby application and tell Phusion Passenger to reload it. You can also run Deploy on other instances so they can, for example, update their configuration to accommodate the newly deployed app.

> **Note**
>
> Setup includes Deploy; it runs the Deploy recipes after setup is complete.

Undeploy

This event occurs when you delete an app or run an Undeploy command to remove an app from a set of application server instances. The specified instances run recipes to remove all application versions and perform any required cleanup.

Shutdown

This event occurs after you direct OpsWorks Stacks to shut an instance down but before the associated Amazon EC2 instance is actually terminated. OpsWorks Stacks runs recipes to perform cleanup tasks such as shutting down services.

If you have attached an Elastic Load Balancing load balancer to the layer and enabled support for connection draining, OpsWorks Stacks waits until connection draining is complete before triggering the Shutdown event.

After triggering a Shutdown event, OpsWorks Stacks allows Shutdown recipes a specified amount of time to perform their tasks, and then stops or terminates the Amazon EC2 instance. The default Shutdown timeout value is 120 seconds. If your Shutdown recipes might require more time, you can edit the layer configuration to change the timeout value. For more information on instance Shutdown, see Stopping an Instance.

> **Note**
>
> Rebooting an instance does not trigger any lifecycle events.

For more discussion about the Deploy and Undeploy app commands, see Deploying Apps.

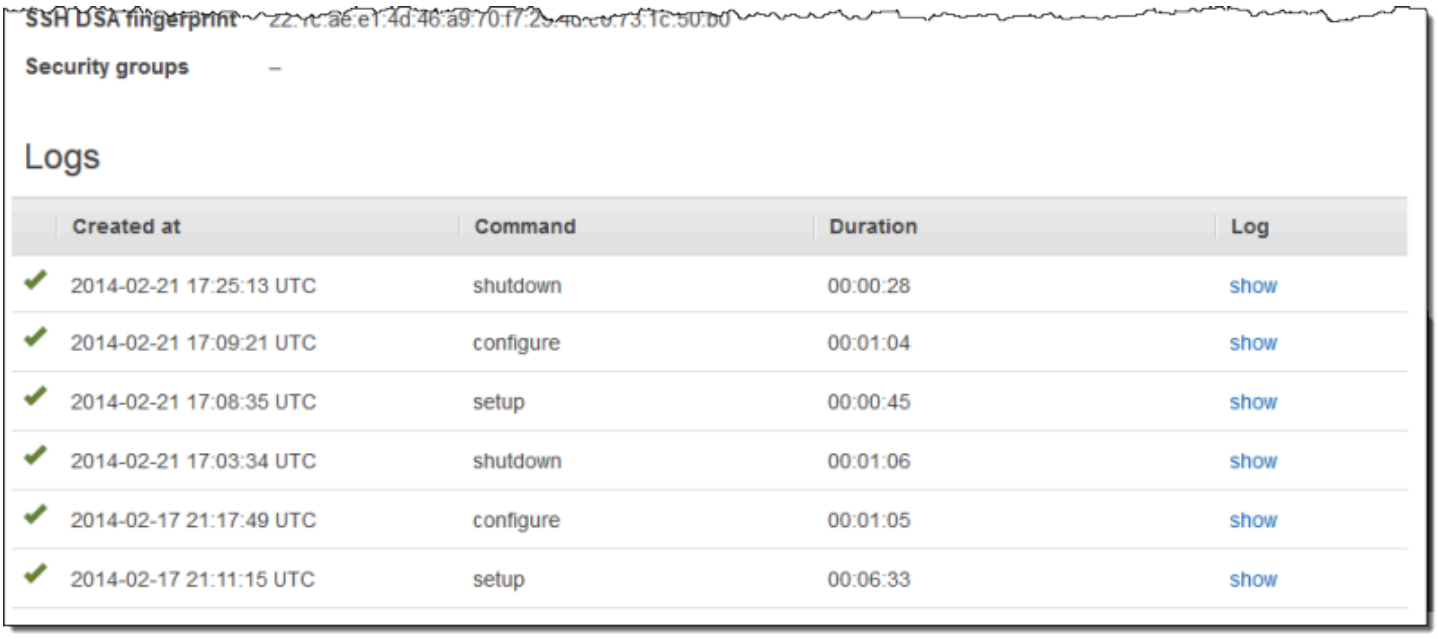
After a started instance has finished booting, the remaining startup sequence is as follows:

1. OpsWorks Stacks runs the instance's built-in Setup recipes, followed by any custom Setup recipes.
2. OpsWorks Stacks runs the instance's built-in Deploy recipes, followed by any custom Deploy recipes.

   The instance is now online.
3. OpsWorks Stacks triggers a Configure event on all instances in the stack, including the newly started instance.

   OpsWorks Stacks runs the instances' built-in Configure recipes, followed by any custom Configure recipes.

> **Note**
>
> To see the lifecycle events that have occurred on a particular instance, go to the **Instances** page and click the instance's name to open its details page. The list of events is in the **Logs** section at the bottom of the page. You can click **show** in the **Log** column to examine the Chef log for an event. It provides detailed information about how the event was handled, including which recipes were run. For more information on how to interpret Chef logs, see Chef Logs.



For each lifecycle event, OpsWorks Stacks installs a set of stack configuration and deployment attributes on each instance that contains the current stack state and, for Deploy events, information about the deployment. The attributes include information about what instances are available, their IP addresses, and so on. For more information, see Stack Configuration and Deployment Attributes.

Note

Starting or stopping a large number of instances at the same time can rapidly generate a large number of Configure events. To avoid unnecessary processing, OpsWorks Stacks responds to only the last event. That event's stack configuration and deployment attributes contain all the information required to update the stack's instances for the entire set of changes. This eliminates the need to also process the earlier Configure events. OpsWorks Stacks labels the unprocessed Configure events as **superseded**.

Automatically Running Recipes

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

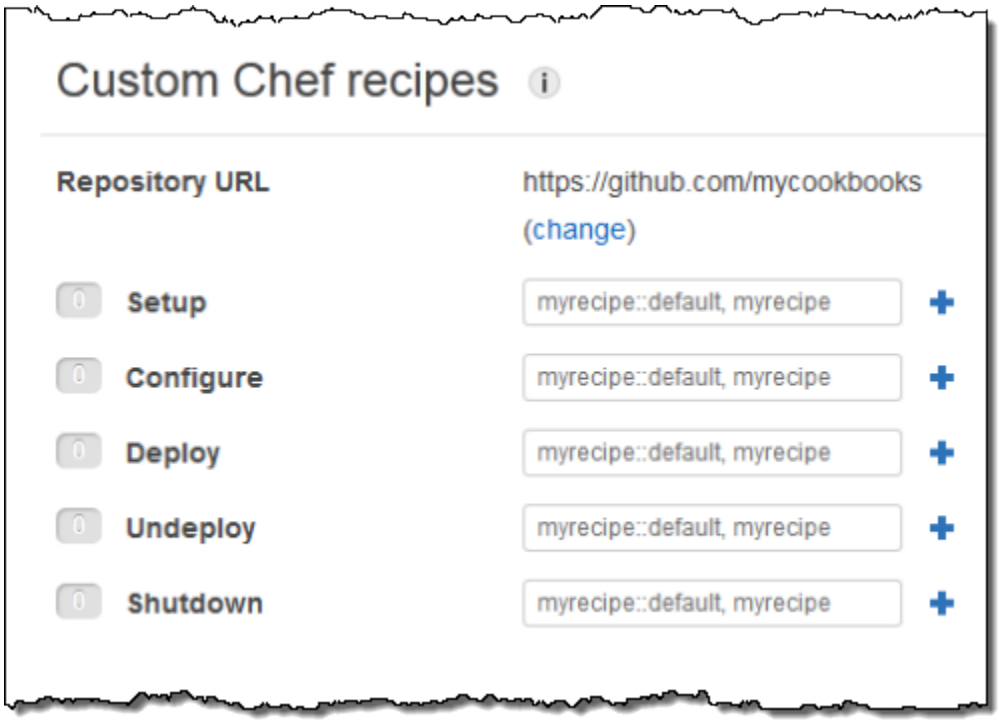
migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Each layer has a set of built-in recipes assigned to each lifecycle event, although some layers lack Undeploy recipes. When a lifecycle event occurs on an instance, OpsWorks Stacks runs the appropriate set of recipes for the associated layer.

If you have installed custom cookbooks, you can have OpsWorks Stacks run some or all of the recipes automatically by assigning each recipe to a layer's lifecycle event. After an event occurs, OpsWorks Stacks runs the specified custom recipes after the layer's built-in recipes.

To assign custom recipes to layer events

1. On the **Layers** page, for the appropriate layer, click **Recipes** and then click **Edit**. If you haven't yet enabled custom cookbooks, click **configure cookbooks** to open the stack's **Settings** page. Toggle **Use custom Chef Cookbooks** to **Yes**, and provide the cookbook's repository information. Then click **Save** and navigate back to the edit page for the **Recipes** tab. For more information, see Installing Custom Cookbooks.
2. On the **Recipes** tab, enter each custom recipe in the appropriate event field and click **+** to add it to the list. Specify a recipe as follows: *`cookbook`*::*`somerecipe`* (omit the `.rb` extension).



When you start a new instance, OpsWorks Stacks automatically runs the custom recipes for each event, after it runs the standard recipes.

> **Note**
>
> Custom recipes execute in the order that you enter them in the console. An alternative way to control execution order is to implement a meta recipe that executes the recipes in the correct order.

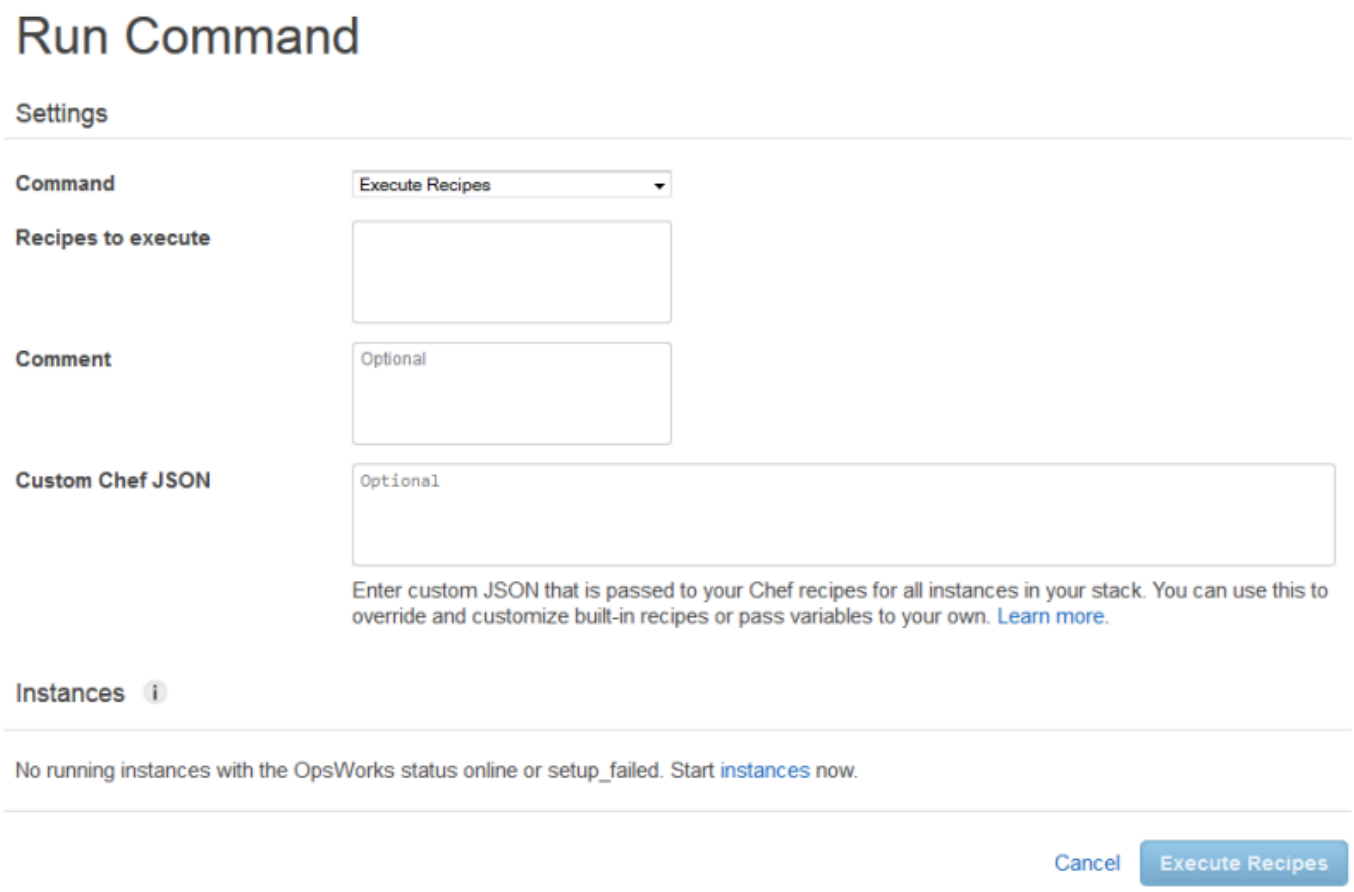
Manually Running Recipes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Although recipes typically are run automatically in response to lifecycle events, you can manually run recipes at any time on any or all stack instances. This feature is typically used for tasks that don't map well to a lifecycle event, such as backing up instances. To run a custom recipe manually, it must be in one of your custom cookbooks, but it does not have to be assigned to a lifecycle event. When you run a recipe manually, OpsWorks Stacks installs the same `deploy` attributes that it does for a Deploy event.

To manually run recipes on stack instances

1. On the **Stack** page, click **Run command**. For **Command**, select **Execute Recipes**.



2. Enter the recipes to be run in the **Recipes to execute** box by using the standard *cookbookname*::*recipename* format. Use commas to separate multiple recipes; they will run in the order that you list them.
3. Optionally, use the **Custom Chef JSON** box to add a custom JSON object that defines custom attributes that will be merged into the stack configuration and deployment attributes that are installed on the instances. For more information about using custom JSON objects, see Using Custom JSON and Overriding Attributes.
4. Under **Instances**, select the instances on which OpsWorks Stacks should run the recipes.

When a lifecycle event occurs, the OpsWorks Stacks agent receives a command to run the associated recipes. You can manually run these commands on a particular instance by using the appropriate stack command or by using the agent CLI's run_command command. For more information on how to use the agent CLI, see OpsWorks Stacks Agent CLI.

Resource Management

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The **Resources** page enables you to use your account's Elastic IP address, Amazon EBS volume, or Amazon RDS instance resources in an OpsWorks Stacks stack. You can use **Resources** to do the following:

- Register a resource with a stack, which allows you to attach the resource to one of the stack's instances.
- Attach a resource to one of the stack's instances.
- Move a resource from one instance to another.
- Detach a resource from an instance. The resource remains registered and can be attached to another instance.
- Deregister a resource. An unregistered resource cannot be used by OpsWorks Stacks, but it remains in your account unless you delete it, and can be registered with another stack.

Note the following constraints:

- You cannot attach registered Amazon EBS volumes to Windows instances.
- The Resources page manages standard, PIOPS, Throughput Optimized HDD, Cold HDD, or General Purpose (SSD) Amazon EBS volumes, but not RAID arrays.
- Amazon EBS volumes must be xfs formatted.

  OpsWorks Stacks does not support other file formats, such as ext4. For more information on preparing Amazon EBS volumes, see Making an Amazon EBS Volume Available for Use.
- You can't attach an Amazon EBS volume to—or detach it from—a running instance.

You can operate only on offline instances. For example, you can register an in-use volume with a stack and attach it to an offline instance, but you must stop the original instance and detach the volume before starting the new instance. Otherwise, the start process will fail.

- All registered resources are managed solely in OpsWorks. This can override resource lifecycle properties, such as `DeleteOnTermination` for EC2 volumes.
- You can attach an Elastic IP address to and detach it from a running instance.

  You can operate on online or offline instances. For example, you can register an in-use address and assign it to a running instance, and OpsWorks Stacks will automatically reassign the address.
- To register and deregister resources, your IAM policy must grant permissions for the following actions:

Amazon EBS Volumes	Elastic IP Addresses	Amazon RDS Instances
RegisterVolume	RegisterElasticIp	RegisterRdsDbInstance
UpdateVolume	UpdateElasticIp	UpdateRdsDbInstance
DeregisterVolume	DeregisterElasticIp	DeregisterRdsDbInstance

The Manage permissions level grants permissions for all of these actions. To prevent a Manage user from registering or deregistering particular resources, edit their IAM policy to deny permissions for the appropriate actions. For more information, see Security and Permissions.

Topics

- Registering Resources with a Stack
- Attaching and Moving Resources
- Detaching Resources
- Deregistering Resources

Registering Resources with a Stack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Amazon EBS volumes or Elastic IP addresses must be registered with a stack before you can attach them to instances. When OpsWorks Stacks creates resources for a stack, they are automatically registered with that stack. If you want to use externally-created resources, you must explicitly register them. Note the following:

- You can register a resource with only one stack at a time.
- When you delete a stack, OpsWorks Stacks deregisters all resources.

Topics

- Registering Amazon EBS Volumes with a Stack
- Registering Elastic IP Addresses with a Stack
- Registering Amazon RDS Instances with a Stack

Registering Amazon EBS Volumes with a Stack

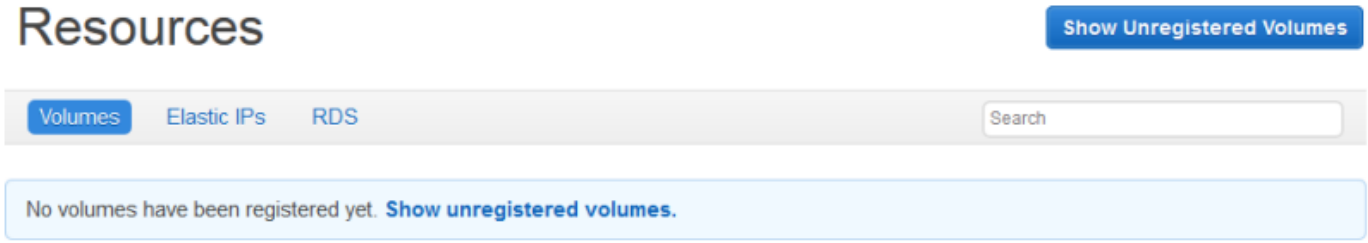
> **Note**
>
> This resource can be used only with Linux stacks. Although you can register an Amazon EBS volume with a Windows stack, you cannot attach it to an instance.

You can use the **Resources** page to register an Amazon EBS volume with a stack, subject to the following constraints:

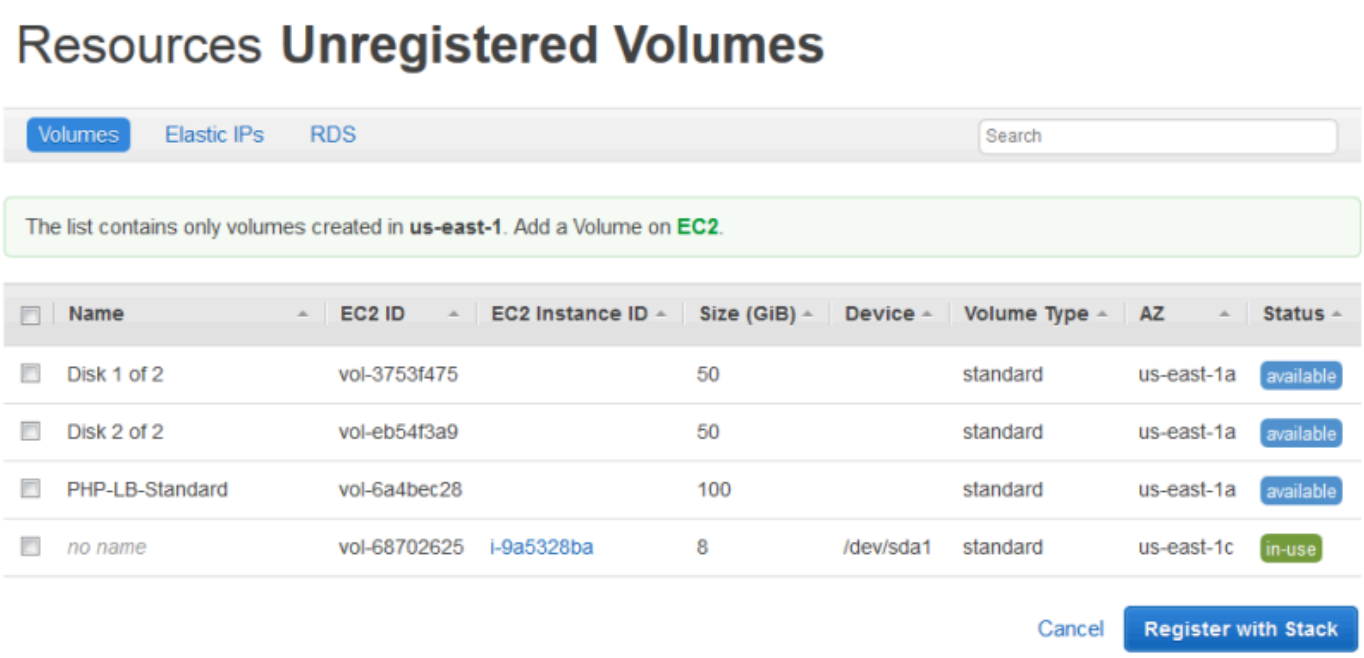
- Attached, non-root Amazon EBS volumes must be standard, Throughput Optimized HDD, Cold HDD, PIOPS, or General Purpose (SSD), but not a RAID array. For information about maximum and minimum volume sizes, see EBS Volumes in this guide.
- Volumes must be XFS formatted.
- OpsWorks Stacks does not support other file formats, such as fourth extended file system (ext4), for non-root Amazon EBS volumes. For more information about preparing Amazon EBS volumes, see Making an Amazon EBS Volume Available for Use. Note that the example in that topic describes how to create an ext4-based volume, but you can follow the same steps for XFS based volumes.

To register an Amazon EBS volume

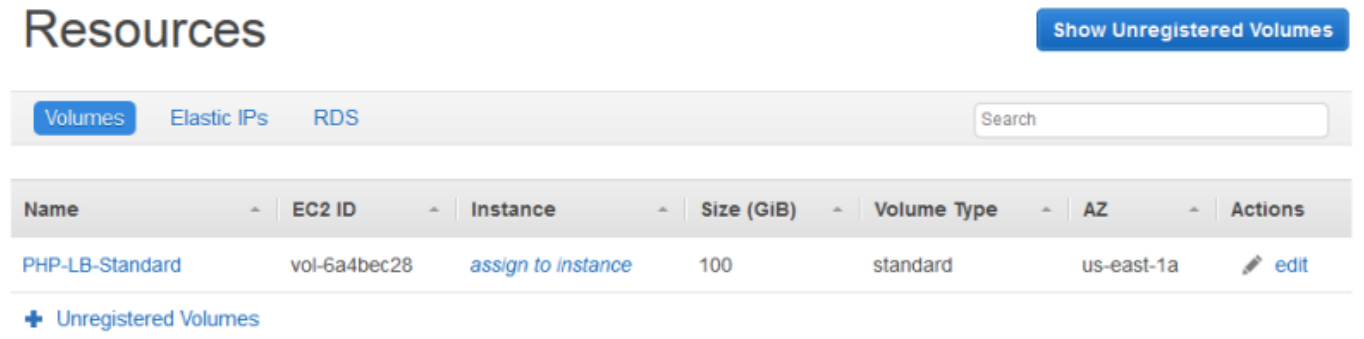
1. Open the desired stack and click **Resources** in the navigation pane.
2. Click **Volumes** to display the available Amazon EBS volumes. Initially, the stack has no registered volumes, as shown in the following illustration.



3. Click **Show Unregistered Volumes** to display the Amazon EBS volumes in your account that are in the stack's region and if applicable, the stack's VPC. The **Status** column indicates whether the volumes are available for use. **Volume Type** indicates whether the volume is standard (`standard`), General Purpose SSD (gp2), PIOPS (`io1`, followed by the IOPS per disk value in parentheses), Throughput Optimized HDD (`st1`), or Cold HDD (`sc1`).



4. Select the appropriate volumes and click **Register to Stack**. The **Resources** page now lists the newly registered volumes.



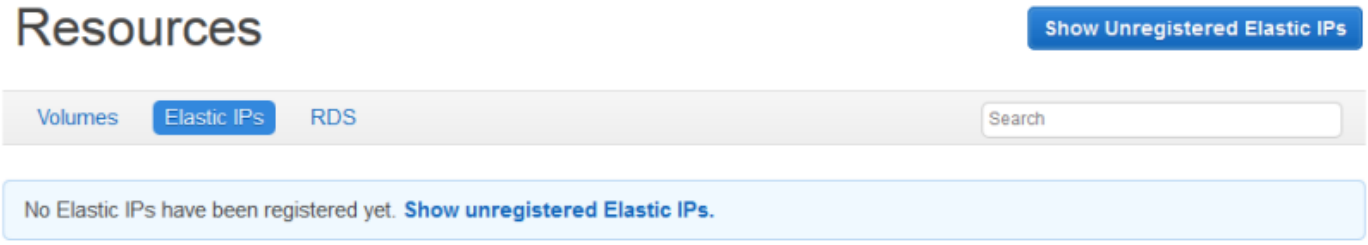
   To register additional volumes, click **Show Unregistered Volumes** or **+ Unregistered Volumes** and repeat this procedure.

Registering Elastic IP Addresses with a Stack

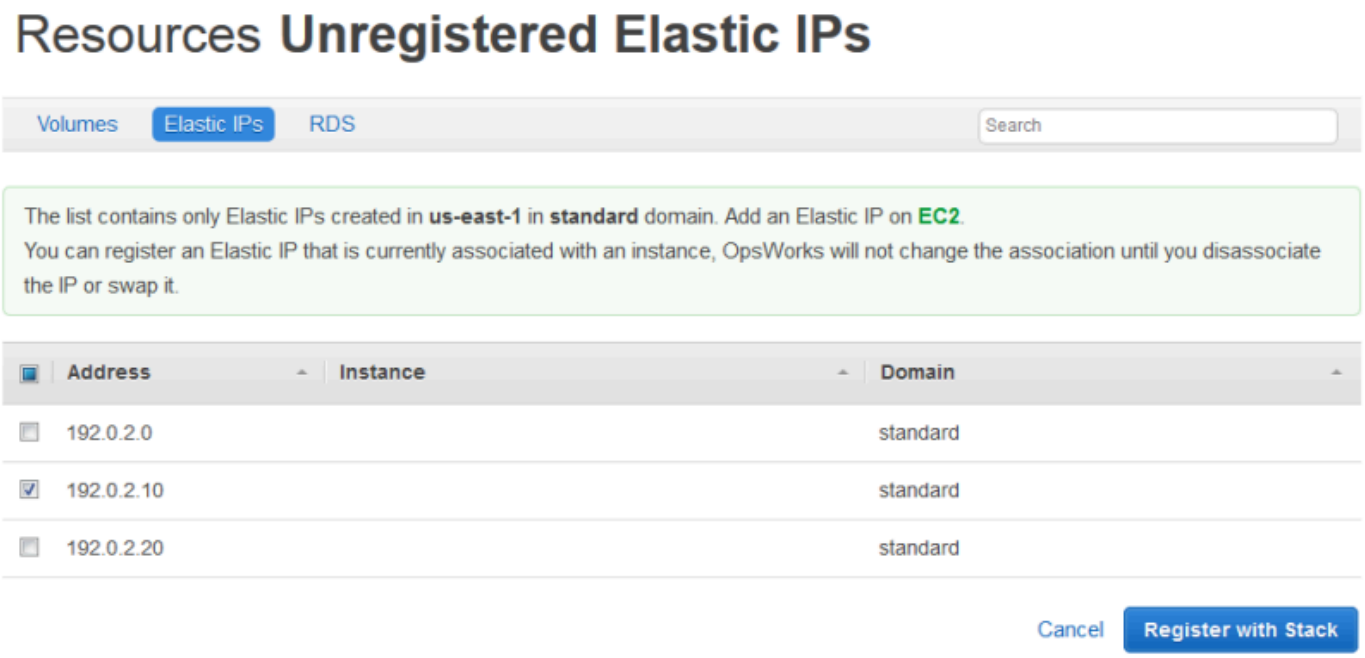
Use the following procedure to register Elastic IP addresses.

To register an Elastic IP address

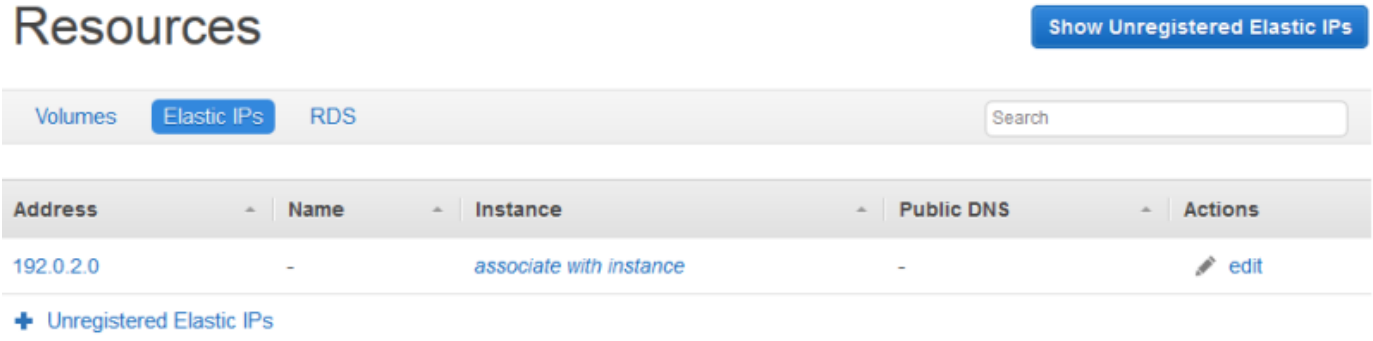
1. Open the stack's **Resources** page and click **Elastic IPs** to display the available Elastic IP addresses. Initially, the stack has no registered addresses, as shown in the following illustration.



2. Click **Show Unregistered Elastic IPs** to display the available Elastic IP addresses in your account that are in the stack's region.



3. Select the appropriate addresses and click **Register to Stack**. This returns you to the **Resources** page, which now lists the newly registered addresses.



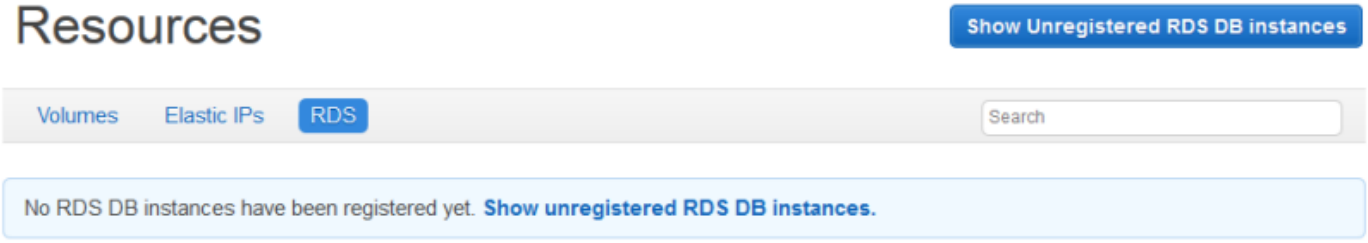
To register additional addresses, click **Show Unregistered Elastic IPs** or **+ Unregistered Elastic IPs** and repeat this procedure.

Registering Amazon RDS Instances with a Stack

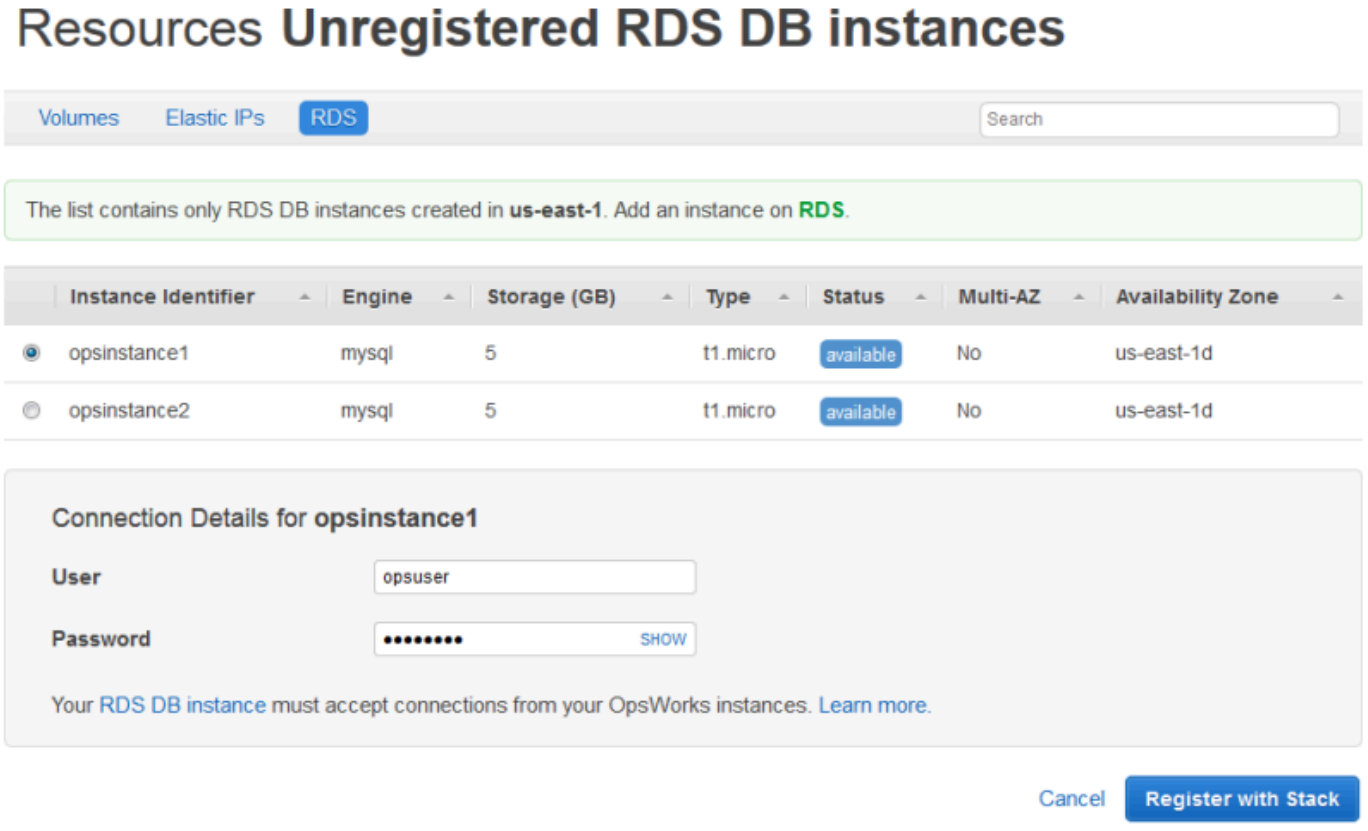
Use the following procedure to register Amazon RDS instances.

To register an Amazon RDS instance

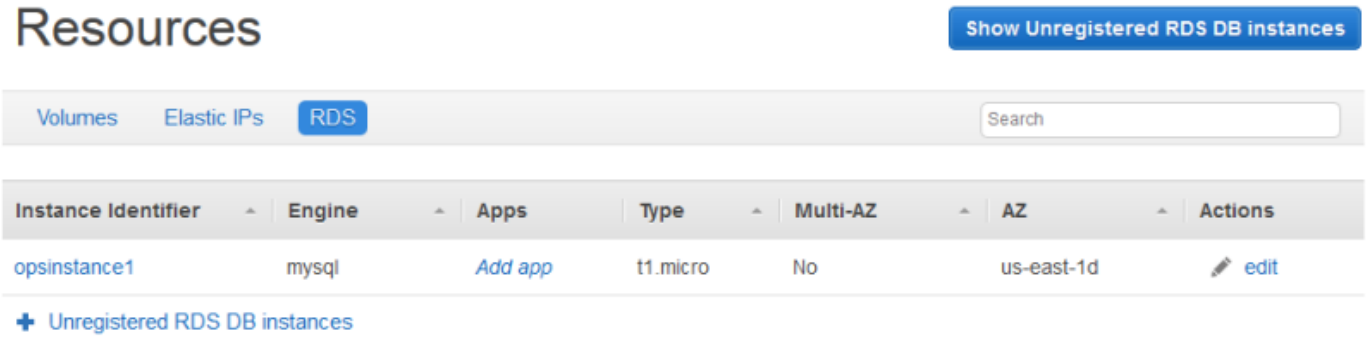
1. Open the stack's **Resources** page and click **RDS** to display the available Amazon RDS instances. Initially, the stack has no registered instances, as shown in the following illustration.



2. Click **Show Unregistered RDS DB instances** to display the available Amazon RDS instances in your account that are in the stack's region.



3. Select the appropriate instance, enter its master user and master password values for **User** and **Password**, and click **Register to Stack**. This returns you to the **Resources** page, which now lists the newly registered instance.



> **Important**
>
> You must ensure that the user and password that you use to register the Amazon RDS instance correspond to a valid user and password. If they do not, your applications will not be able connect to the instance.

To register additional addresses, click **Show Unregistered RDS DB instances** or **+ Unregistered RDS DB instances** and repeat this procedure. For more information about how to use Amazon RDS instances with OpsWorks Stacks, see Amazon RDS Service Layer.

> **Note**
>
> You can also register Amazon RDS instances through the **Layers** page. For more information, see Amazon RDS Service Layer.

Attaching and Moving Resources

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

After you register a resource with a stack, you can attach it to one of the stack's instances. You can also move an attached resource from one instance to another. Note the following:

- When you attach or move Amazon EBS volumes, the instances involved in the operation must be offline. If the instance you are interested in is not on the **Resources** page, go to the **Instances** page and stop the instance. After it has stopped, you can return to the **Resources** page and attach or move the resource.
- When you attach or move Elastic IP addresses, the instances can be online or offline.
- If you delete an instance, any attached resources remain registered with the stack. You can then attach the resource to another instance or, if you no longer need it, deregister the resource.

Topics

- Assigning Amazon EBS Volumes to an Instance
- Associating Elastic IP Addresses with an Instance
- Attaching Amazon RDS Instances to an App

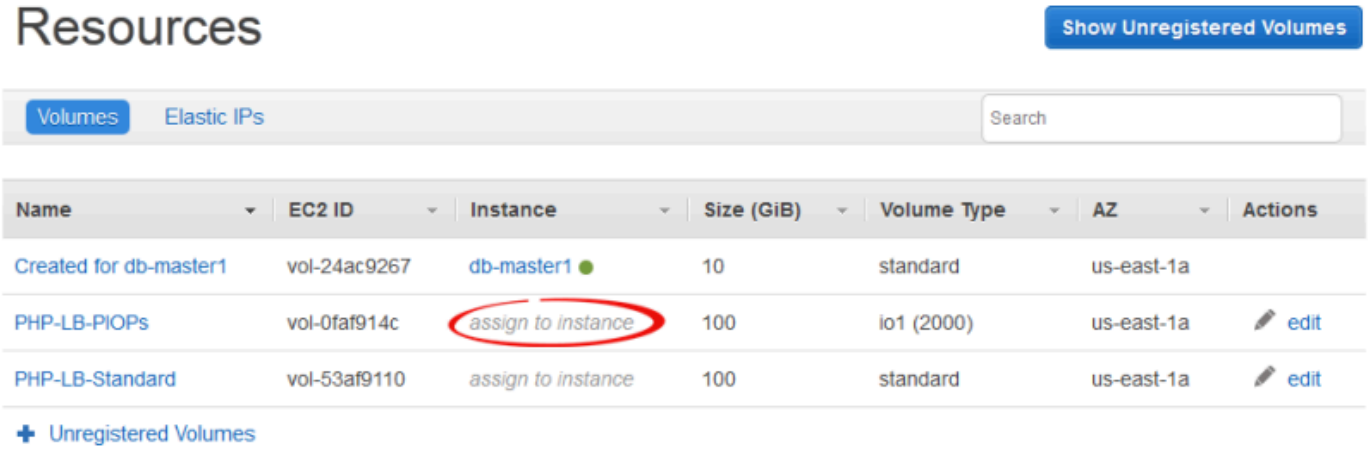
Assigning Amazon EBS Volumes to an Instance

> **Note**
>
> You cannot assign Amazon EBS volumes to Windows instances.

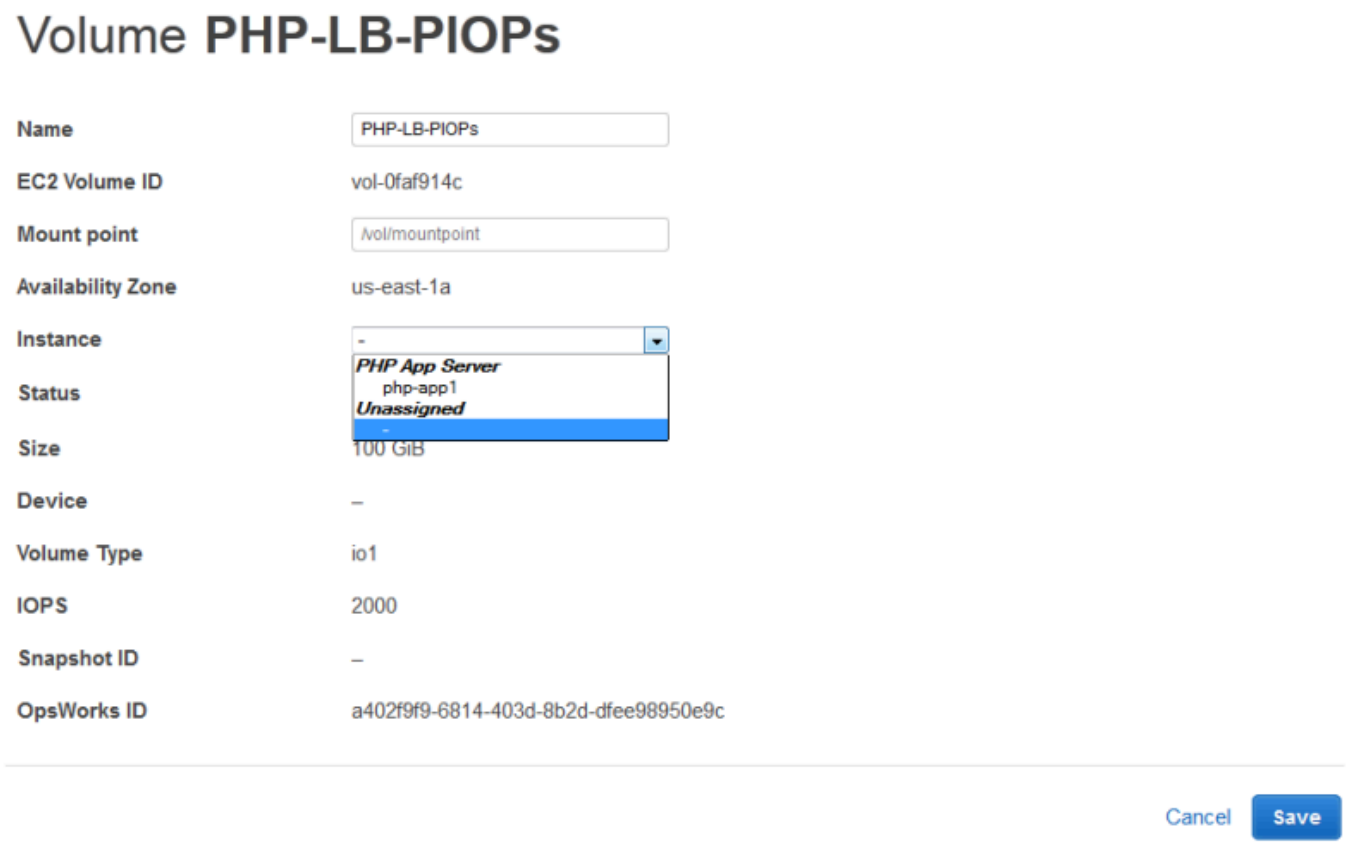
You can assign a registered Amazon EBS volume to an instance and move it from one instance to another, but both instances must be offline.

To assign an Amazon EBS volume to an instance

1. On the Resources page, click **assign to instance** in the appropriate volume's **Instance** column.



2. On the volume's details page, select the appropriate instance, specify the volume's name and mount point, and click **Save** to attach the volume to the instance.



> **Important**
>
> If you have assigned an external in-use volume to your instance, you must use the Amazon EC2 console, API, or CLI to unassign it from the original instance or the start process will fail.

You can also use the details page to move an assigned Amazon EBS volume to another instance in the stack.

To move an Amazon EBS volume to another instance

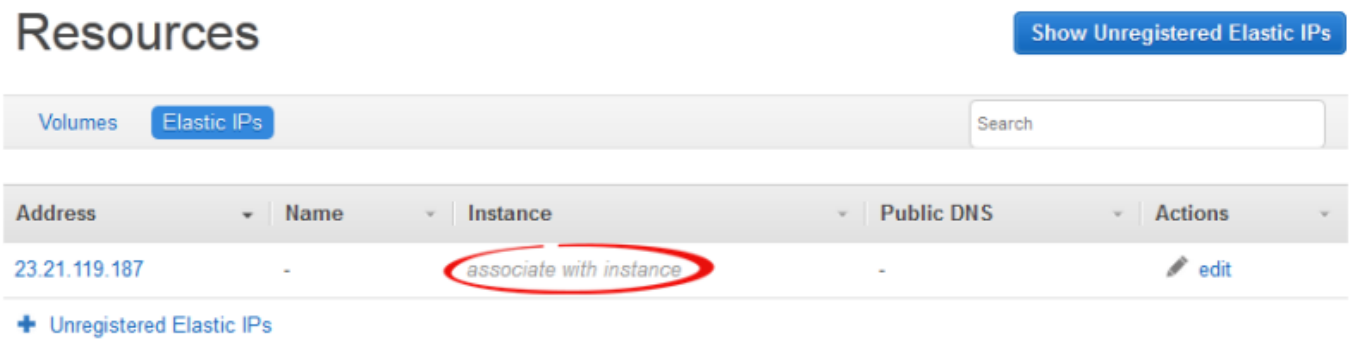
1. Ensure that both instances are in the offline state.
2. On the **Resources** page, click **Volumes** and then click **edit** in the volume's **Actions** column.
3. Do one of the following:
   - To move the volume to another instance in the stack, select the appropriate instance from the **Instance** list and click **Save**.
   - To move the volume to an instance in another stack, deregister the volume, register the volume with the new stack, and attach it to the news instance.

Associating Elastic IP Addresses with an Instance

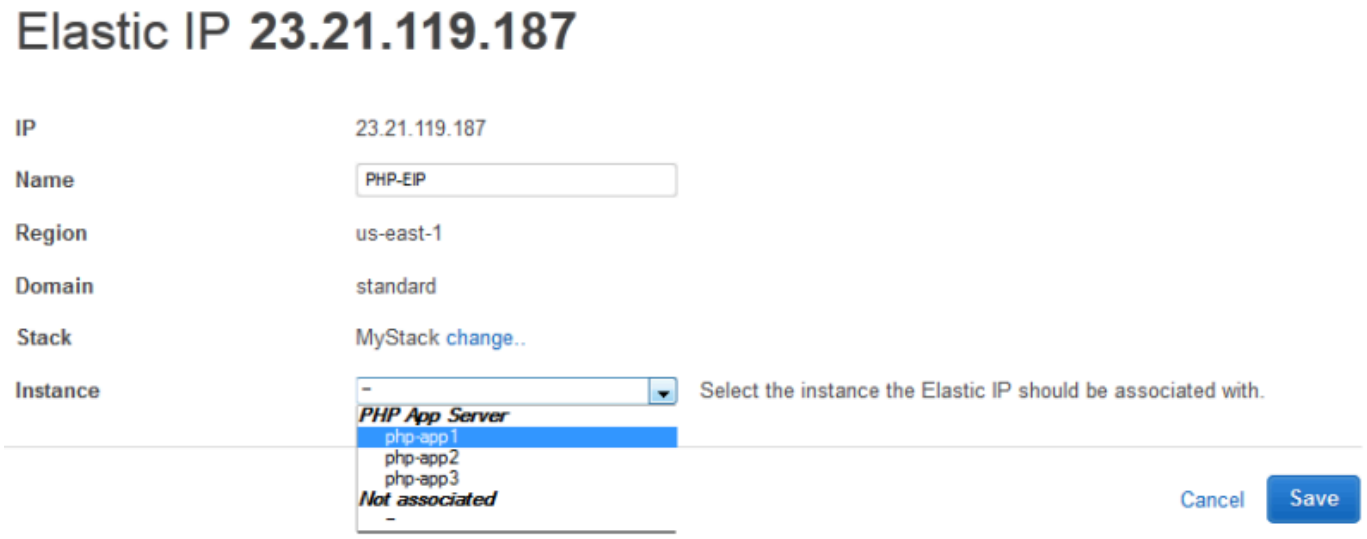
You can associate a registered Elastic IP address with an instance and move it from one instance to another, including instances in other stacks. The instances can be either online or offline.

To associate an Elastic IP address with an instance

1. On the **Resources** page, click **associate with instance** in the appropriate address's **Instance** column.



2. On the address's details page, select the appropriate instance, specify the address's name, and click **Save** to associate the address with the instance.



> **Note**
>
> If the Elastic IP address is currently associated with another online instance, OpsWorks Stacks automatically reassigns the address to the new instance.

You can also use the details page to move an associated Elastic IP address to another instance.

To move an Elastic IP address to another instance

1. On the **Resources** page, click **Elastic IPs** and click **edit** in the address's **Actions** column.
2. Do one of the following:

   - To move the address to another instance in the stack, select the appropriate instance from the **Instance** list and click **Save**.
   - To move the address to an instance in another stack, click **change** in the **Stack** settings to see a list of the available stacks. Select a stack from the **Stack** list and an instance from the **Instance** list. Then click **Save**.



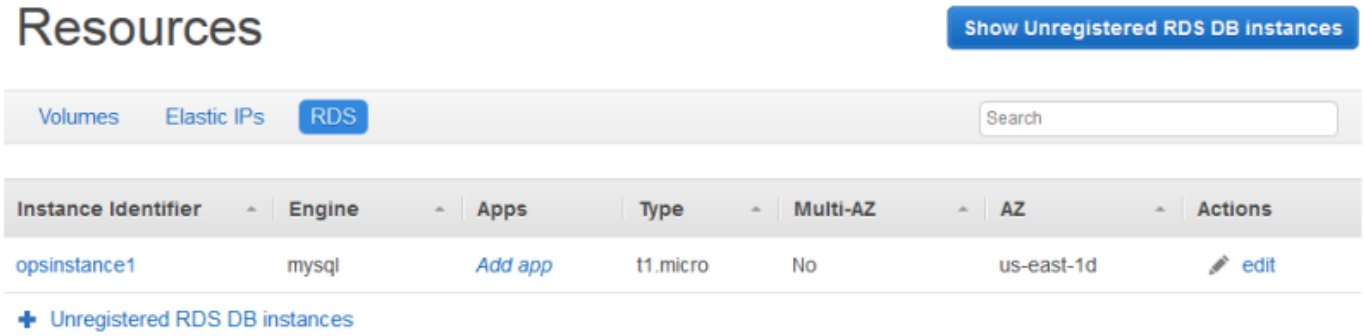
After you attach or move an address, OpsWorks Stacks triggers a Configure lifecycle event to notify the stack's instances of the change.

Attaching Amazon RDS Instances to an App

You can attach an Amazon RDS instance to one or more apps.

To attach an Amazon RDS instance to an app

1. On the **Resources** page, click **Add app** in the appropriate instance's **Apps** column.



2. Use the **Add App** page to attach the Amazon RDS instance. For more information, see Adding Apps.

Because an Amazon RDS can be attached to multiple apps, there is no special procedure for moving the instance from one app to another. Just edit the first app to remove the RDS instance or edit the second app to add the RDS instance. For more information, see Editing Apps.

Detaching Resources

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

When you no longer need an attached resource, you can detach it. This resource remains registered with the stack and can be attached elsewhere.

Topics

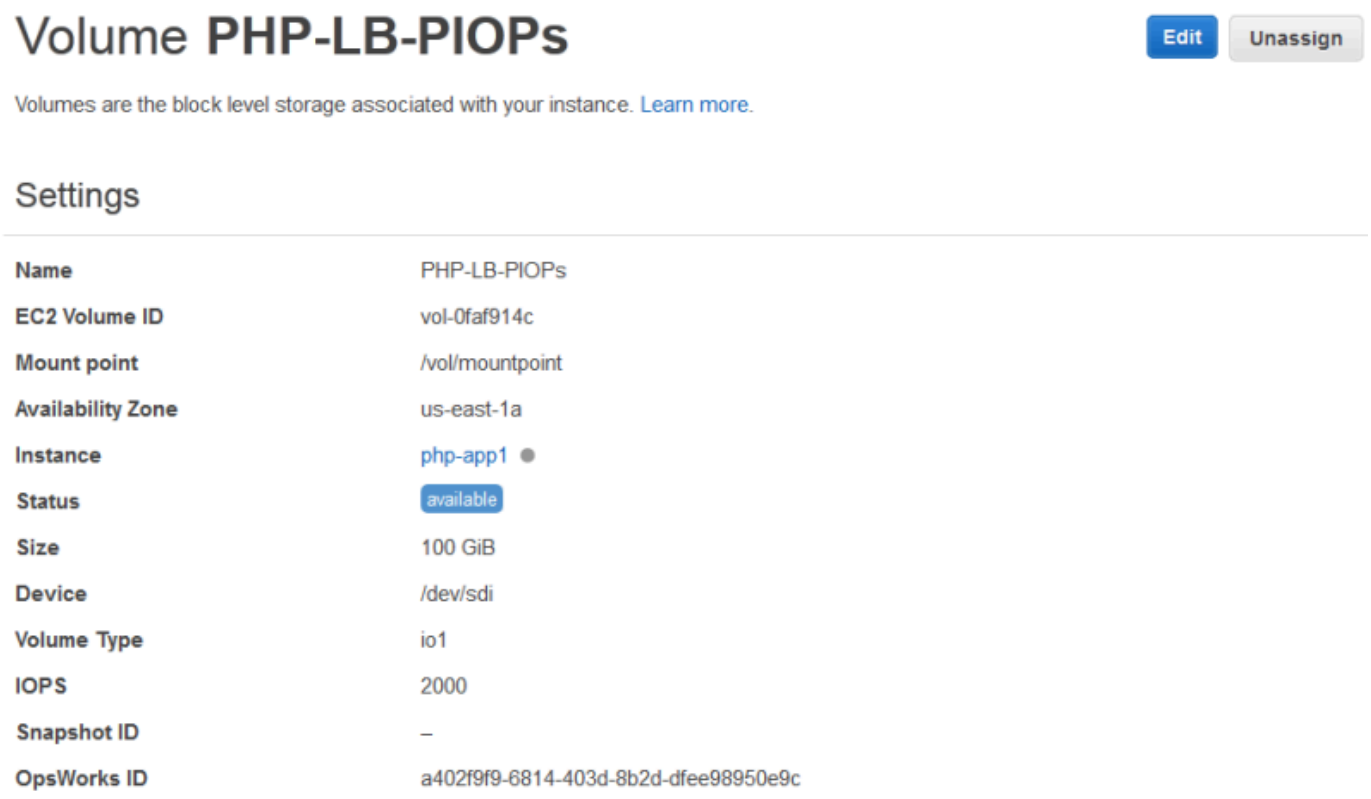
- Unassigning Amazon EBS Volumes
- Disassociating Elastic IP Addresses
- Detaching Amazon RDS Instances

Unassigning Amazon EBS Volumes

Use the following procedure to unassign an Amazon EBS volume from its instance.

To unassign an Amazon EBS volume

1. Ensure that the instance is in the offline state.
2. On the **Resources** page, click **Volumes** and click volume name.
3. On the volume's details page, click **Unassign**.

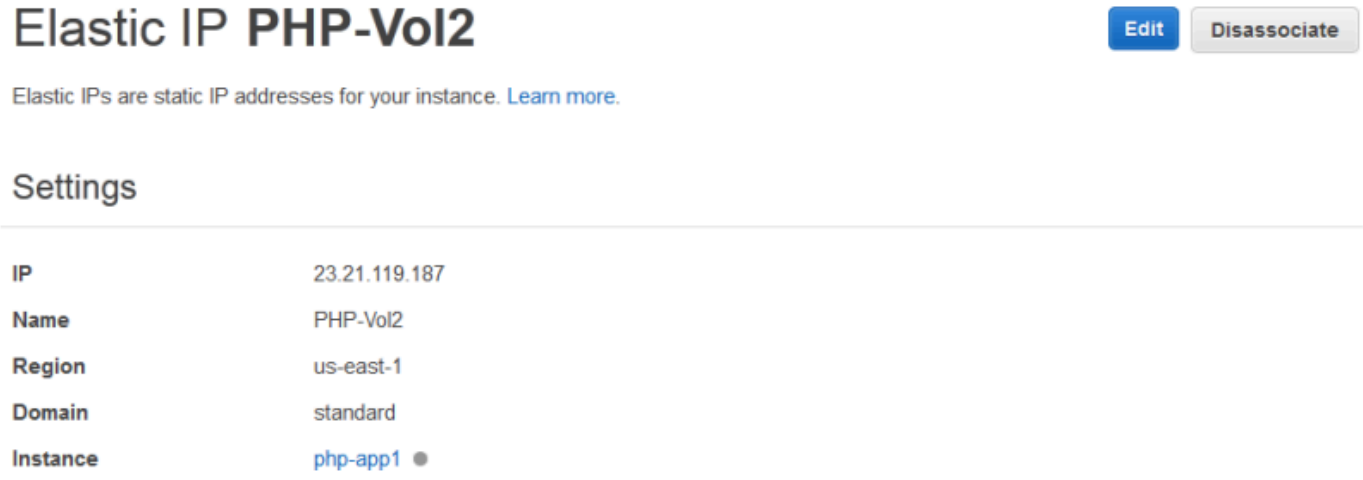


Disassociating Elastic IP Addresses

Use the following procedure to disassociate an Elastic IP address from its instance.

To disassociate an Elastic IP address

1. On the **Resources** page, click **Elastic IPs** and click **edit** in the address's **Actions** column.
2. On the address's details page, click **Disassociate**.



After you disassociate an address, OpsWorks Stacks triggers a Configure lifecycle event to notify the stack's instances of the change.

Detaching Amazon RDS Instances

Use the following procedure to detach an Amazon RDS from an app.

To detach an Amazon RDS instance

1. On the **Resources** page, click **RDS** and click the appropriate app in the **Apps** column.
2. Click **Edit** and edit the app configuration to detach the instance. For more information, see Editing Apps.

> **Note**
>
> This procedure detaches an Amazon RDS from a single app. If the instance is attached to multiple apps, you must repeat this procedure for each app.

Deregistering Resources

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

If you no longer need to have a resource registered with a stack, you can deregister it. Deregistration does not delete the resource from your account; it remains there and can be registered with another stack or used outside OpsWorks Stacks. If you want to delete the resource entirely, you have two options:

- If an Elastic IP or Amazon EBS resource is attached to an instance, you can delete the resource when you delete the instance.

Go to the **Instances** page, click **delete** in the instance's **Actions** column, and then select **Delete instance's EBS volumes** or **Delete the instance's Elastic IP**.

- Deregister the resource and then use the Amazon EC2 or Amazon RDS console, API, or CLI to delete it.

Topics

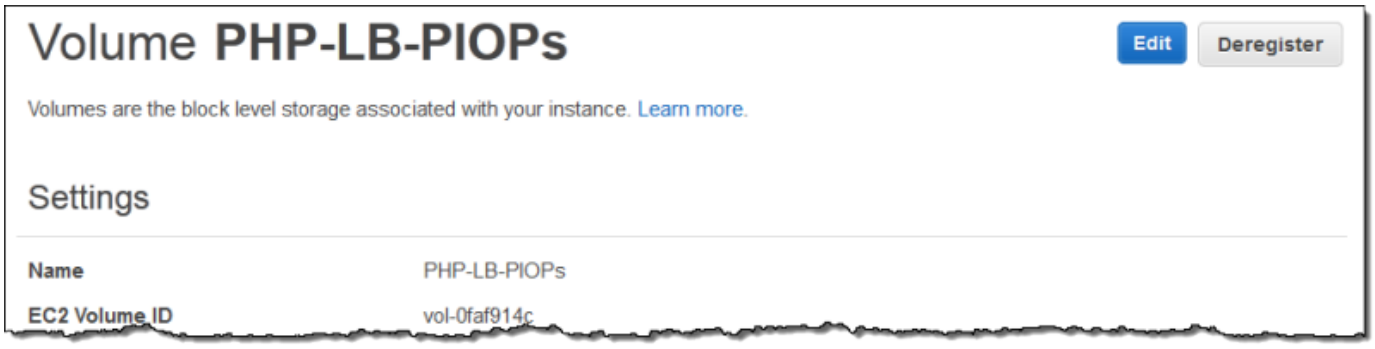
- Deregistering Amazon EBS Volumes
- Deregistering Elastic IP Addresses
- Deregistering Amazon RDS Instances

Deregistering Amazon EBS Volumes

Use the following procedure to deregister an Amazon EBS volume.

To deregister an Amazon EBS volume

1. If the volume is attached to an instance, unassign it, as described in Unassigning Amazon EBS Volumes.
2. On the **Resources** page, click the volume name in the **Name** column.
3. On the volume's details page, click **Deregister**.

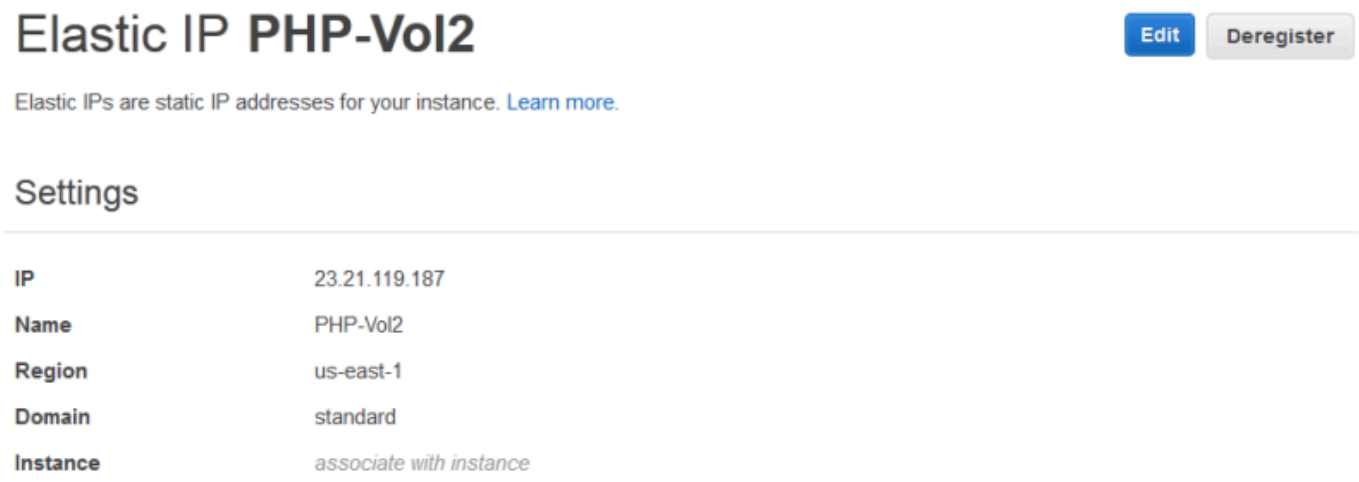


Deregistering Elastic IP Addresses

Use the following procedure to deregister an Elastic IP address.

To deregister an Elastic IP address

1. If the address is associated with an instance, disassociate it, as described in Disassociating Elastic IP Addresses.
2. On the **Resources** page, click **Elastic IPs** and then click the IP address in the **Address** column.
3. On the address's details page, click **Deregister**.



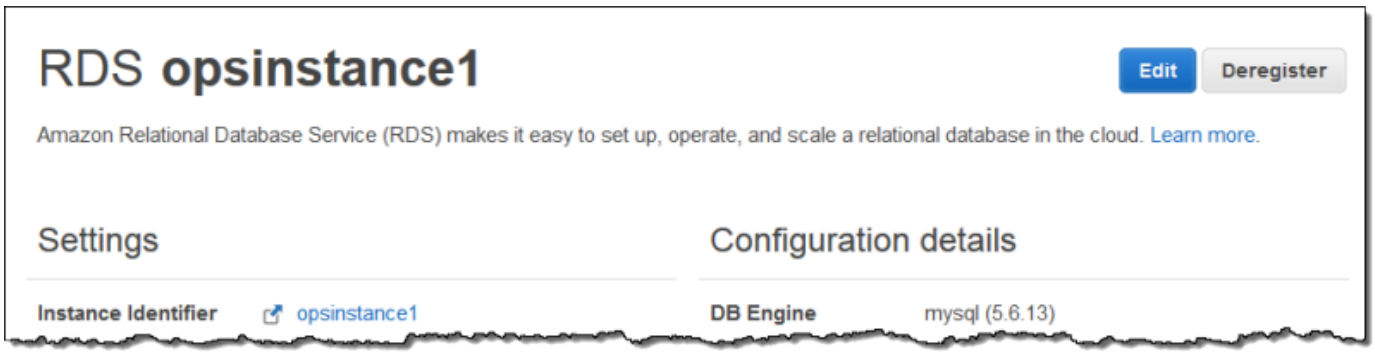
> **Note**
>
> If you simply want to register an Elastic IP address with a different stack, you must deregister it from its current stack and then register it with the new stack. However, you can move an attached Elastic IP address to an instance in another stack directly. For more information, see Attaching and Moving Resources.

Deregistering Amazon RDS Instances

Use the following procedure to deregister an Amazon RDS instance.

To deregister an Amazon RDS instance

1. If the instance is associated with an app, detach it, as described in Detaching Resources.
2. On the **Resources** page, click **RDS** and then instance's name.
3. On the instance's details page, click **Deregister**.



Tags

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Tags can help you group resources in Chef 11.10, Chef 12, and Chef 12.2 stacks, and track the costs of using those resources in AWS Billing and Cost Management.

You can apply tags at the stack and layer level. When you create a tag, you are applying the tag to every resource within the tagged structure. For example, if you apply a tag to a layer, you are applying the tag to every instance, Amazon EBS volume (except the root), or Elastic Load Balancing load balancer in the layer. Tags cannot currently be applied to the root, or default, EBS volume of an instance.

Tags are key-value pairs that you assign to stacks or layers in OpsWorks Stacks. After you create tags, open the Billing and Cost Management console to activate user-defined tags. For more information about how to activate your tags and use them to track and manage the costs of your OpsWorks Stacks resources, see Using Cost Allocation Tags and Activating User-Defined Cost Allocation Tags in the *Billing and Cost Management User Guide*.

Tags work in a way that's similar to custom attributes in OpsWorks Stacks. Tags that you apply to a stack are inherited by each layer in the stack. At the layer level, you can override the values (but not

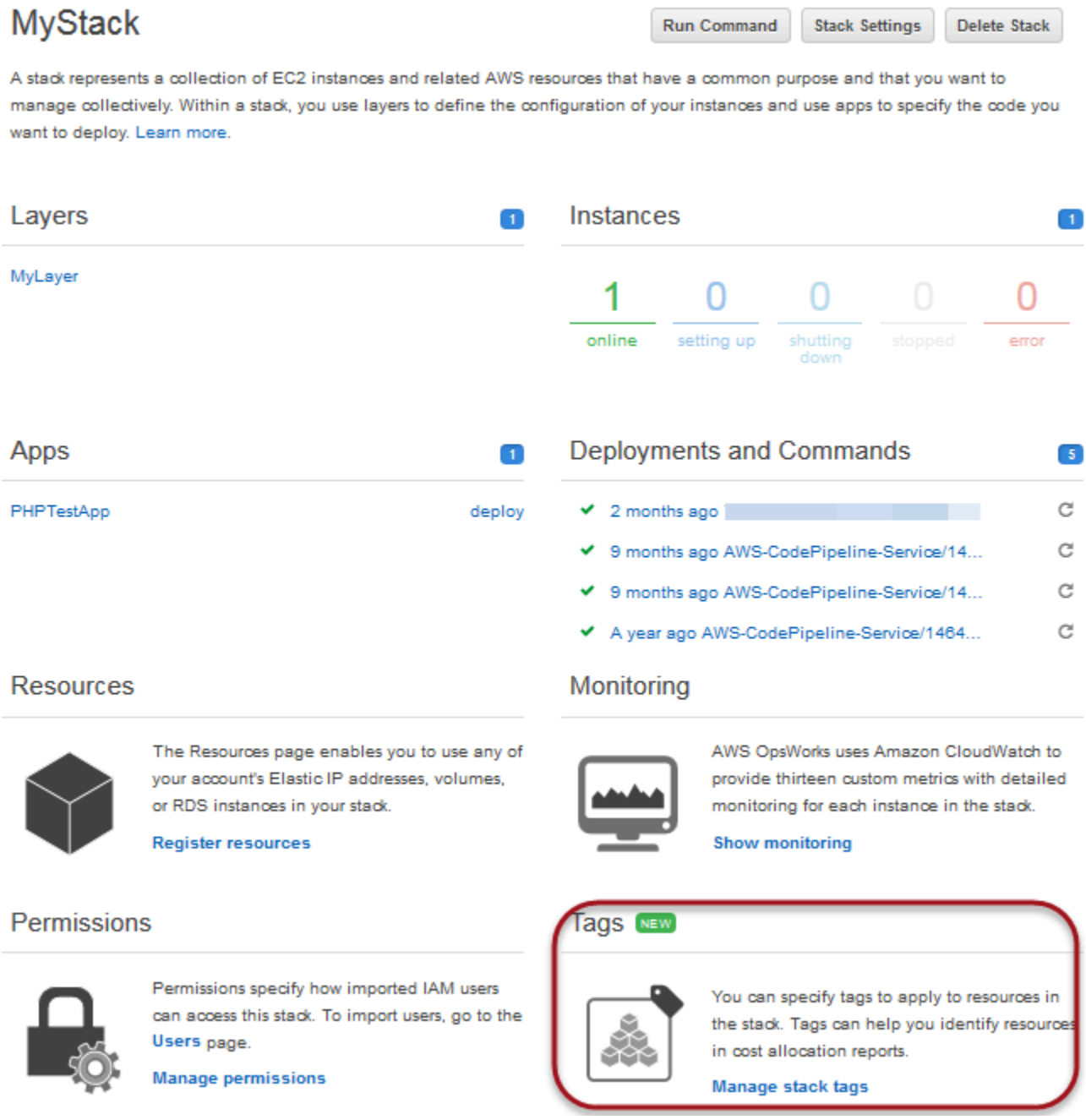
the key names) of inherited tags, and add new layer-specific tags. OpsWorks applies the resulting tag set to all resources in the layer. As you create new or assign existing resources to a layer, the new resources in the layer are tagged with the same set of tags.

Topics

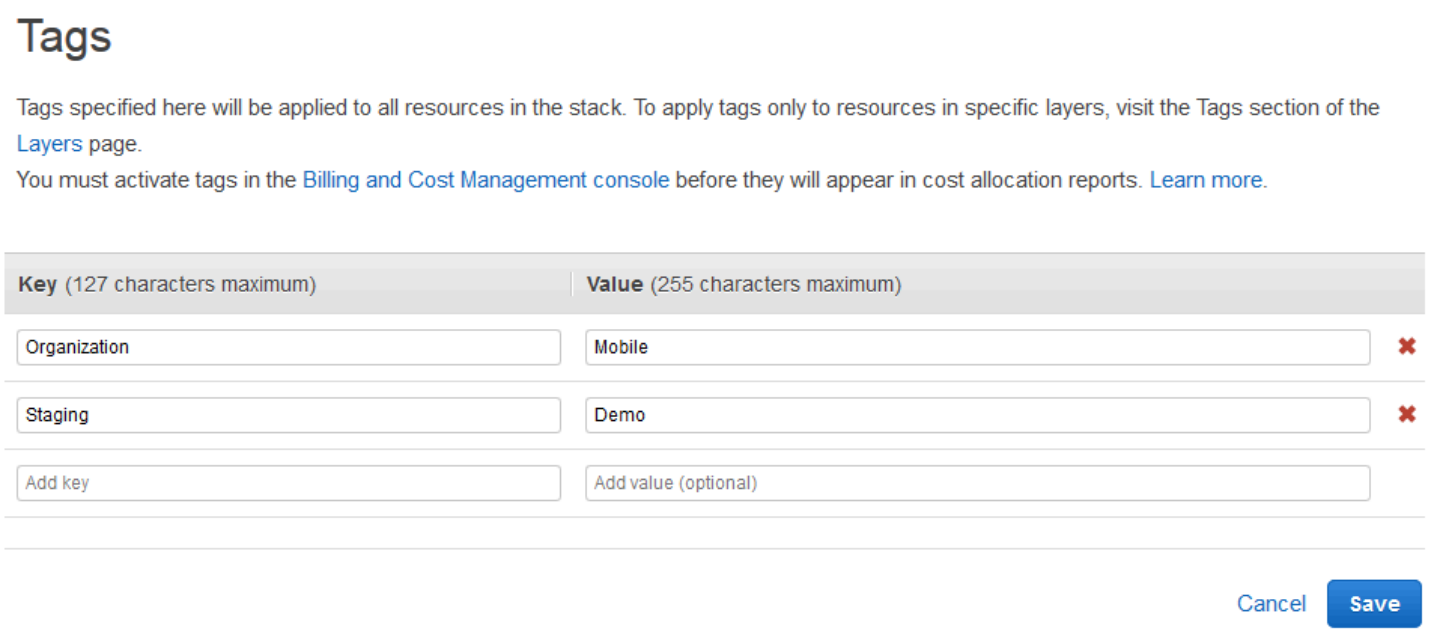
- Setting Tags at the Stack Level
- Setting Tags at the Layer Level
- Managing Tags with the AWS CLI
- Tag Limitations

Setting Tags at the Stack Level

At the stack level, you can add and manage tags by choosing **Tags** on the stack's home page.

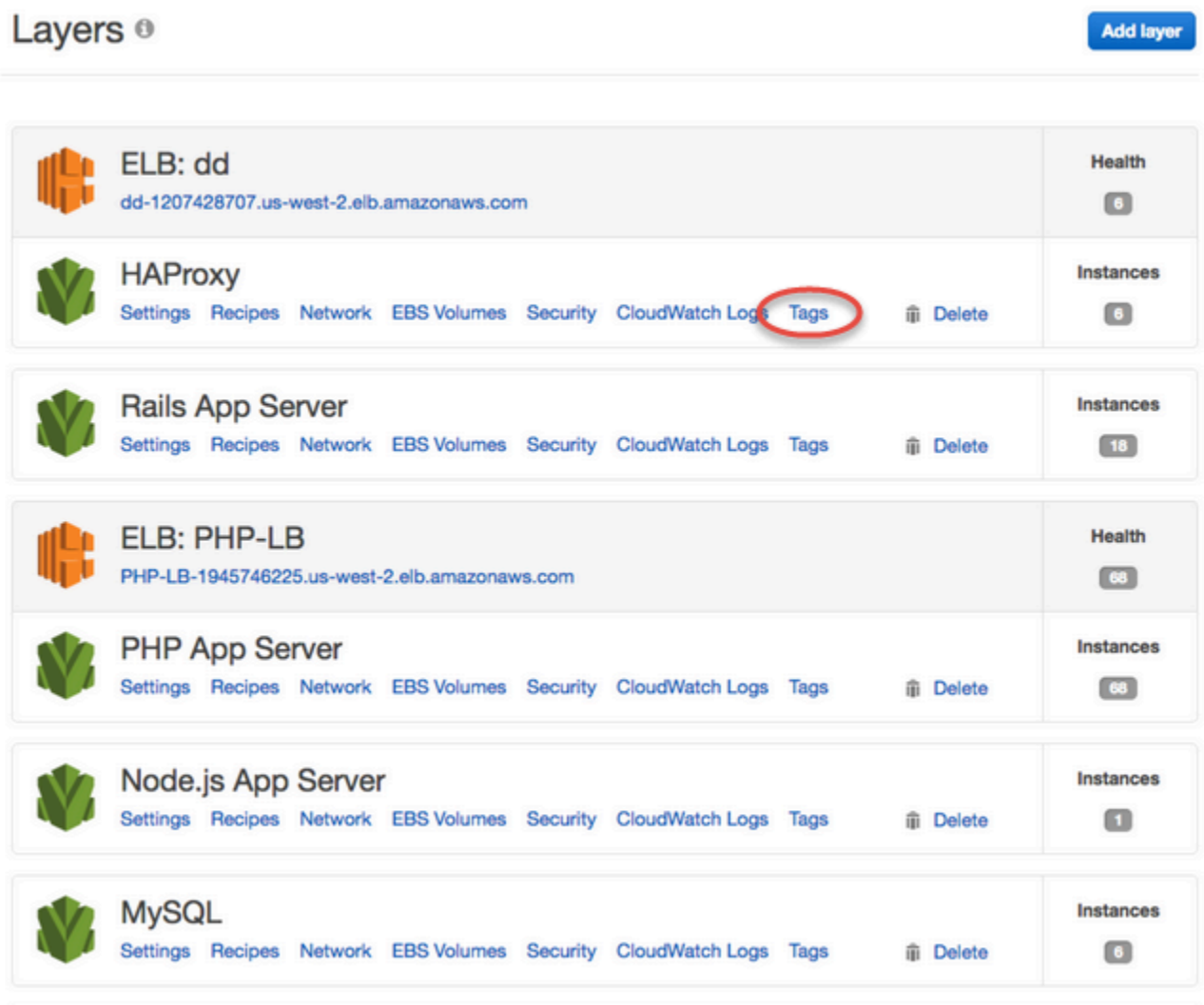


On the **Tags** page, add tags as key-value pairs. The following screenshot shows some example tags. You can delete tags by choosing the red **X** to the right of a key-value pair.

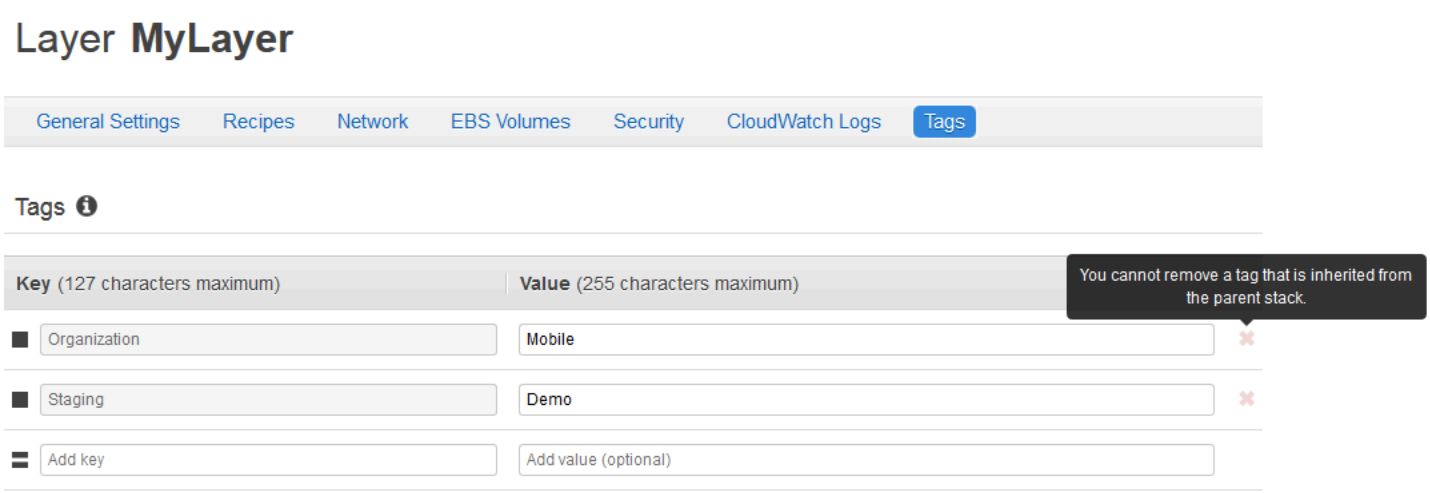


Setting Tags at the Layer Level

At the layer level, set tags by choosing the **Tags** tab. You can find this tab on the **Layers** home page, and the home page for each individual layer.



When you change or add tags at the layer level, be aware that tags that have been added at the parent stack level are inherited by the layer and its resources. While you can change the values of inherited tags, you cannot change the key names, or delete inherited tags. Change the key names or delete tags inherited from the parent stack in stack settings. The following screenshot shows examples of tags inherited from the stack level. Inherited tags are grayed out.



For more information about adding tags to stacks, see Create a New Stack. For more information about adding tags to layers, see Editing an OpsWorks Layer's Configuration.

Managing Tags with the AWS CLI

You can also use AWS CLI commands to add and remove tags at the stack and layer level. For more information about downloading and installing the AWS CLI, see Installing the AWS Command Line Interface. Remember to add the `--region` parameter to your command if the stack that you want to tag is not in your default region. Layer ARNs do not currently appear in the AWS Management Console. To get the ARN of a layer, run the describe-layers command.

To add tags by using the AWS CLI

- At the AWS CLI command prompt, type the following command, replacing *stack_or_layer_ARN* and specifying your key-value pair tags, and then press **Enter**. Double quotation marks are escaped with backslashes.

  ```
  aws opsworks tag-resource --resource-arn stack_or_layer_ARN --tags "{\"key\":
  \"value\",\"key\":\"value\"}"
  ```

  The following is an example.

  ```
  aws opsworks tag-resource --resource-arn arn:aws:opsworks:us-
  east-2:800000000003:stack/500b99c0-ec00-4cgg-8a0d-1000000jjd1b --tags "{\"Stage\":
  \"Production\",\"Organization\":\"Mobile\"}"
  ```

To remove tags by using the AWS CLI

- At the AWS CLI command prompt, type the following, and then press **Enter**.

  ```
  aws opsworks untag-resource --resource-arn stack_or_layer_ARN --tag-keys "[\"key\",
  \"key\"]"
  ```

  To remove tags, you only specify the key of the tag that you want to remove. The following is an example.

  ```
  aws opsworks untag-resource --resource-arn arn:aws:opsworks:us-
  east-2:800000000003:stack/500b99c0-ec00-4cgg-8a0d-1000000jjd1b --tag-keys "[\"Stage
  \",\"Organization\"]"
  ```

  > **Note**
  >
  > You cannot remove inherited tags (tags that were added at the parent stack level) from a layer. Remove inherited tags from the stack instead.

Tag Limitations

Keep the following limitations in mind when you create tags.

- OpsWorks Stacks limits the number of user-defined tags at the stack and layer level to 40, including user-defined tags inherited from a parent level. This leaves 10 available slots for default tags that are prepended with `opsworks:`, and tags that are set by other AWS processes. A maximum of 50 tags is allowed on a resource, including both user-defined and default tags that are created by AWS.
- Tag keys cannot start with **`aws:`**, **`opsworks:`** or **`rds:`**. Do not use **`name`** or **`Name`** as a tag key, because **`Name`** is reserved by OpsWorks Stacks.
- A key can be a maximum of 127 characters, and can contain only Unicode letters, numbers, or separators, or the following special characters: `+ - = . _ : / .`
- A value can be a maximum 255 characters, and contain only Unicode letters, numbers, or separators, or the following special characters: `+ - = . _ : / .`

Monitoring

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can monitor your stacks in the following ways.

- OpsWorks Stacks uses Amazon CloudWatch to provide thirteen custom metrics with detailed monitoring for each instance in the stack.
- OpsWorks Stacks integrates with AWS CloudTrail to log every OpsWorks Stacks API call and store the data in an Amazon S3 bucket.
- You can use Amazon CloudWatch Logs to monitor your stack's system, application, and custom logs.

Topics

- Monitoring Stacks using Amazon CloudWatch
- Logging OpsWorks Stacks API Calls with AWS CloudTrail
- Using Amazon CloudWatch Logs with OpsWorks Stacks
- Monitoring Stacks using Amazon CloudWatch Events

Monitoring Stacks using Amazon CloudWatch

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks uses Amazon CloudWatch (CloudWatch) to provide monitoring for stacks.

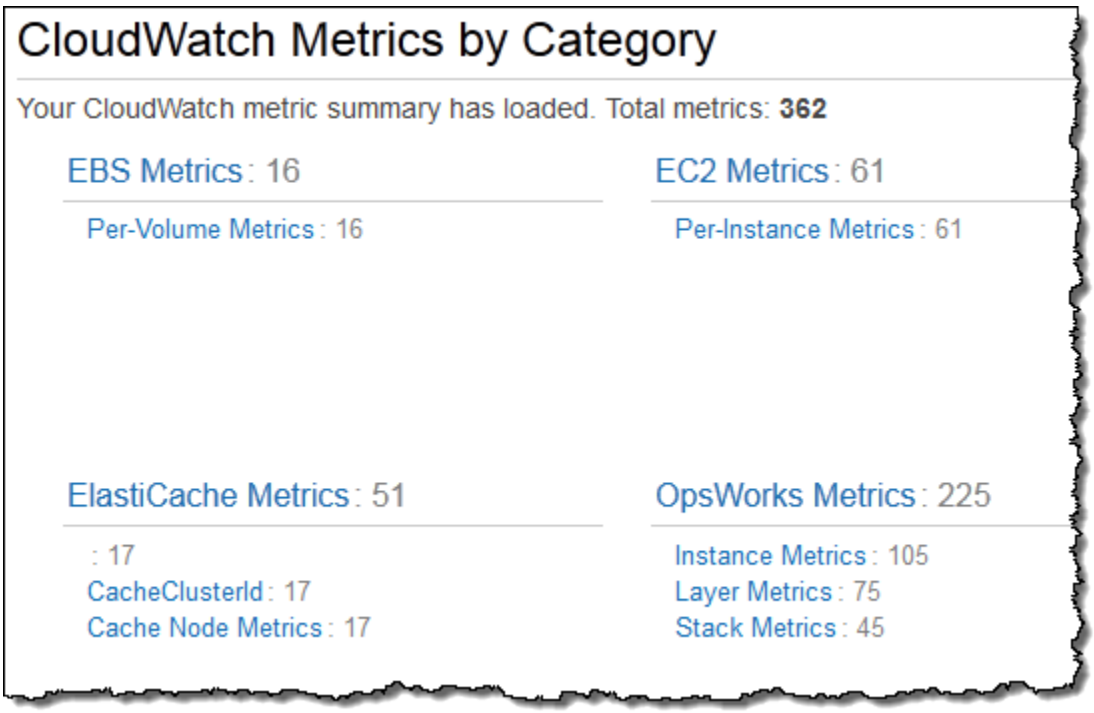
- For Linux stacks, OpsWorks Stacks supports thirteen custom metrics to provide detailed monitoring for each instance in the stack and summarizes the data for your convenience on the **Monitoring** page.
- For Windows stacks, you can monitor standard Amazon EC2 metrics for your instances with the CloudWatch console.

  The **Monitoring** page does not display Windows metrics.

The **Monitoring** page displays metrics for an entire stack, a layer, or an instance. OpsWorks Stacks metrics are distinct from Amazon EC2 metrics. You can also enable additional metrics through the CloudWatch console, but they typically require additional charges. You can also view the underlying data on the CloudWatch console, as follows:

To view OpsWorks custom metrics in CloudWatch

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. On the navigation bar, select the stack's region.
3. In the navigation pane, choose **Metrics**.
4. In OpsWorks Metrics, choose **Instance Metrics**, **Layer Metrics**, or **Stack Metrics**.



Note

OpsWorks Stacks collects metrics by running a process on each instance (the instance agent). Because CloudWatch collects metrics differently, using the hypervisor, the values in the CloudWatch console might differ slightly from the corresponding values on the **Monitoring** page in the OpsWorks Stacks console.

You can also use CloudWatch console to set alarms. For more information about how to create alarms, see Creating Amazon CloudWatch Alarms. For a list of CloudWatch custom metrics, see AWS OpsWorks Metrics and Dimensions. For more information, see Amazon CloudWatch.

Topics

- AWS OpsWorks Stacks Metrics
- Dimensions for OpsWorks Stacks Metrics
- Stack Metrics
- Layer Metrics
- Instance Metrics

AWS OpsWorks Stacks Metrics

OpsWorks Stacks sends the following metrics to CloudWatch every five minutes.

CPU Metrics

Metric	Description
`cpu_idle`	The percentage of time that the CPU is idle. Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId. Valid Statistics: `Average`, `Minimum`, `Maximum`, Sum, or `Data Samples`. Unit: None
`cpu_nice`	The percentage of time that the CPU is handling processes with a positive `nice` value, which have a lower scheduling priority. For more information about what this measures, see nice (Unix). Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId. Valid Statistics: `Average`, `Minimum`, `Maximum`, Sum, or `Data Samples`. Unit: None
`cpu_steal`	As AWS allocates hypervisor CPU resources among increasing numbers of instances, virtualization load rises, and can affect how often the hyperviso r can perform requested work on an instance. `cpu_steal` measures the percentage of time that an instance is waiting for the hypervisor to allocate physical CPU resources.

Metric	Description
	Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId. Valid Statistics: `Average`, `Minimum`, `Maximum`, Sum, or `Data Samples`. Unit: None
`cpu_system`	The percentage of time that the CPU is handling system operations. Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId. Valid Statistics: `Average`, `Minimum`, `Maximum`, Sum, or `Data Samples`. Unit: None
`cpu_user`	The percentage of time that the CPU is handling user operations. Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId. Valid Statistics: `Average`, `Minimum`, `Maximum`, Sum, or `Data Samples`. Unit: None

Metric	Description
cpu_waitio	The percentage of time that the CPU is waiting for input/output operations. Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId. Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples. Unit: None

Memory Metrics

Metric	Description
memory_buffers	The amount of buffered memory. Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId. Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples. Unit: None
memory_cached	The amount of cached memory. Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId. Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples. Unit: None

Metric	Description
memory_free	The amount of free memory. Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId. Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples. Unit: None
memory_swap	The amount of swap space. Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId. Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples. Unit: None
memory_total	The total amount of memory. Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId. Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples. Unit: None

Metric	Description
memory_used	The amount of memory in use. Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId. Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples. Unit: None

Load Metrics

Metric	Description
load_1	The load averaged over a one-minute window. Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId. Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples. Unit: None
load_5	The load averaged over a five-minute window. Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId. Valid Statistics: Average, Minimum, Maximum, Sum, or Data Samples. Unit: None
load_15	The load averaged over a 15-minute window.

Metric	Description
	Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId. Valid Statistics: `Average`, `Minimum`, `Maximum`, Sum, or `Data Samples`. Unit: None

Process Metrics

Metric	Description
`procs`	The number of active processes. Valid Dimensions: The IDs of the individual resources for which you are viewing metrics: StackId, LayerId, or InstanceId. Valid Statistics: `Average`, `Minimum`, `Maximum`, Sum, or `Data Samples`. Unit: None

Dimensions for OpsWorks Stacks Metrics

OpsWorks Stacks metrics use the OpsWorks Stacks namespace, and provide metrics for the following dimensions:

Dimension	Description
`StackId`	Average values for a stack.
`LayerId`	Average values for a layer.
`InstanceId`	Average values for an instance.

Stack Metrics

To view a summary of metrics for an entire stack, select a stack in the OpsWorks Stacks **Dashboard** and then click **Monitoring** in the navigation pane. The following example is for a stack with a PHP and a DB layer.



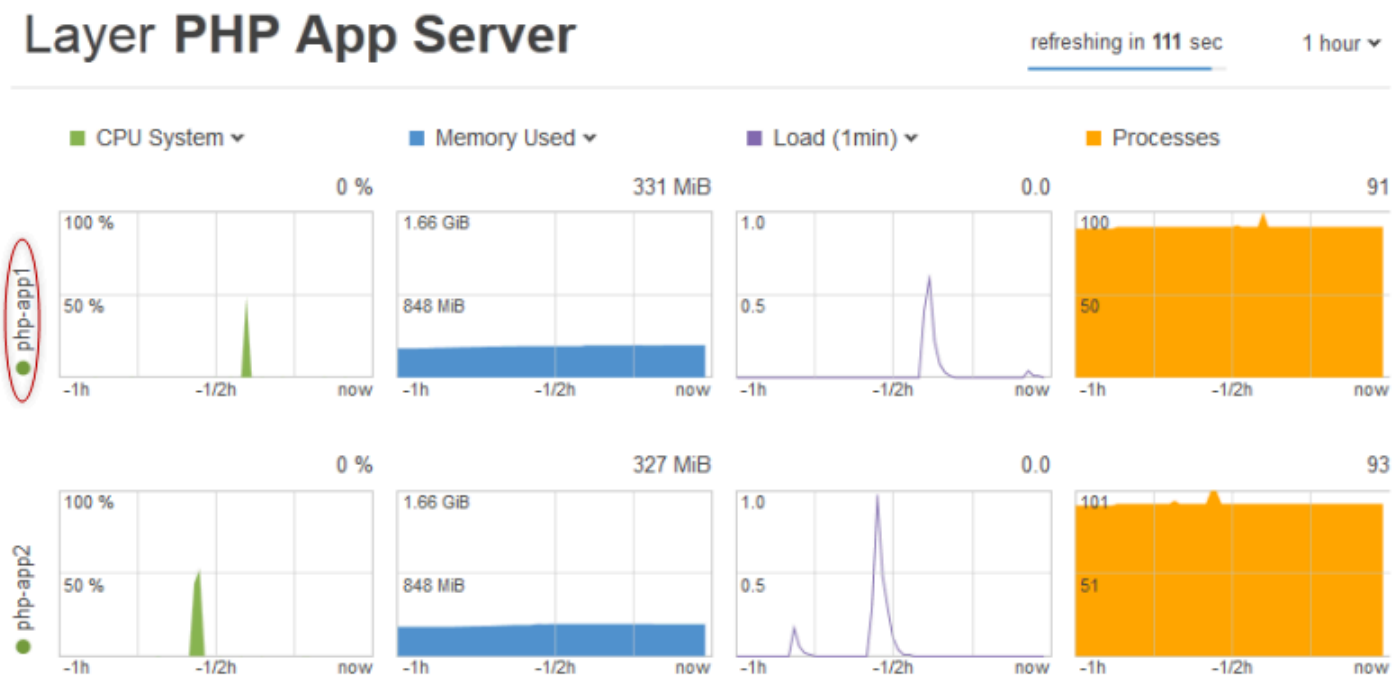
The stack view displays graphs of the four types of metrics for each layer over a specified time period: 1 hour, 8 hours, 24 hours, 1 week, or 2 weeks. Note the following:

- OpsWorks Stacks periodically updates the graphs; the countdown timer at the upper right indicates the time remaining until the next update,
- If a layer has more than one instance, the graphs display average values for the layer.
- You can specify the time period by clicking the list at the upper right and selecting your preferred value.

For each metric type, you can use the list at the top of the graph to select the particular metric that you want to view.

Layer Metrics

To view metrics for a particular layer, click the layer name in the **Monitoring Layers** view. The following example shows metrics for the PHP layer, which has two instances.

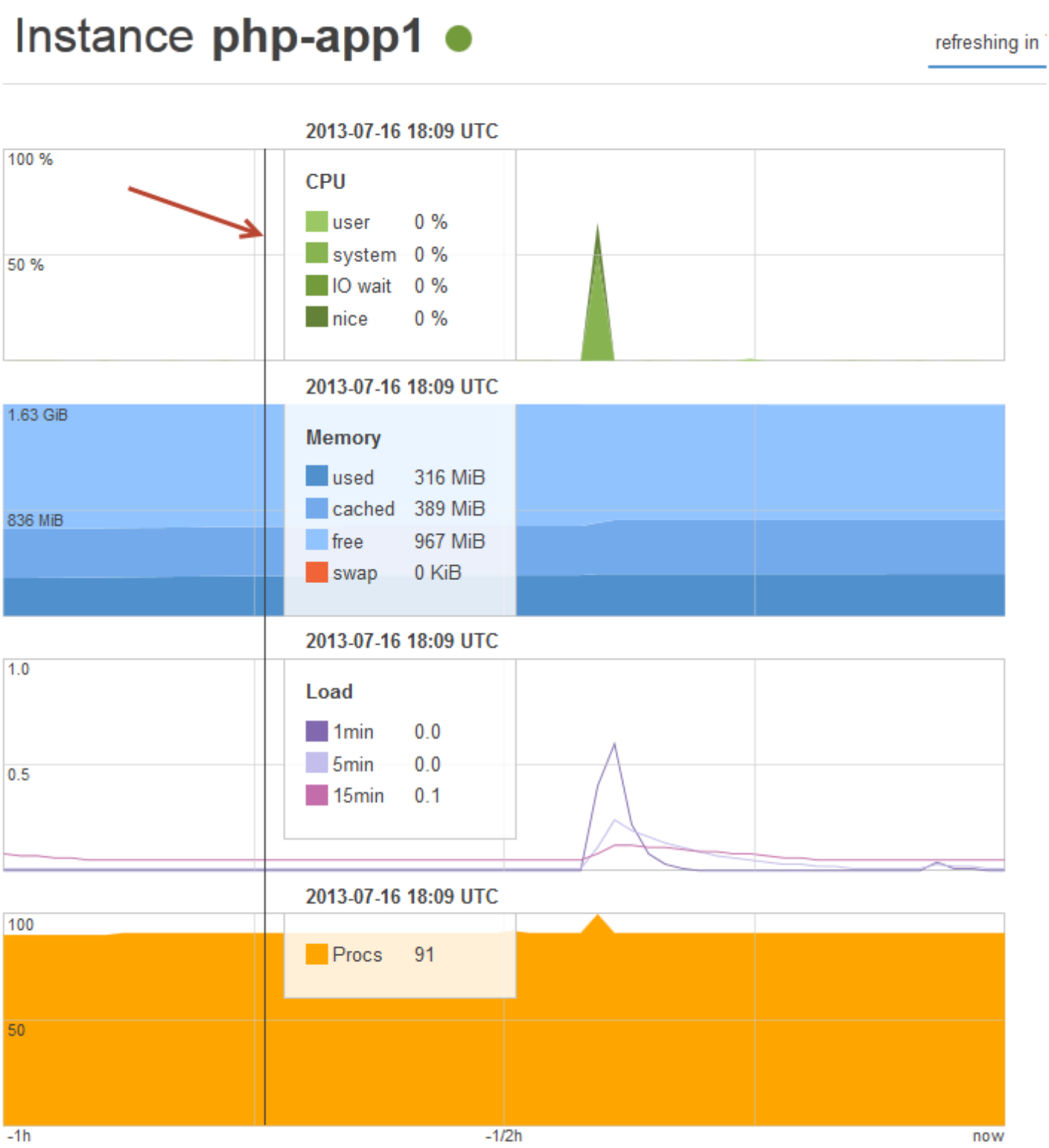


The metric types are the same as for the stack metrics, and for each type, you can use the list at the top of the graph to select the particular metric that you want to view.

> **Note**
>
> You can also display layer metrics by going to the layer's details page and clicking **Monitoring** at the upper right.

Instance Metrics

To view metrics for a particular instance, click the instance name in the layer monitoring view. The following example shows metrics for the PHP layer's **php-app1** instance.


Instance php-app1
refreshing in
2013-07-16 18:09 UTC
100 %
50 %
CPU
user 0 %
system 0 %
IO wait 0 %
nice 0 %
2013-07-16 18:09 UTC
1.63 GiB
836 MiB
Memory
used 316 MiB
cached 389 MiB
free 967 MiB
swap 0 KiB
2013-07-16 18:09 UTC
1.0
0.5
Load
1min 0.0
5min 0.0
15min 0.1
2013-07-16 18:09 UTC
100
50
Procs 91
-1h
-1/2h
now

The graphs summarize all the available metrics for each metric type. To get exact values for a particular point in time, use your mouse to move the slider (indicated by the red arrow in the previous illustration) to the appropriate position.

> **Note**
>
> You can also display instance metrics by going to the instance's details page and choosing **Monitoring** at the upper right.

Logging OpsWorks Stacks API Calls with AWS CloudTrail

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks is integrated with AWS CloudTrail, a service that provides a record of actions taken by an IAM identity, or an AWS service in OpsWorks Stacks. CloudTrail captures all API calls for OpsWorks Stacks as events, including calls from the OpsWorks Stacks console and from code calls to the OpsWorks Stacks APIs. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for OpsWorks Stacks. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to OpsWorks Stacks, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

OpsWorks Stacks Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in OpsWorks Stacks, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for OpsWorks Stacks, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all regions. The trail logs events from all regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All OpsWorks Stacks actions are logged by CloudTrail and are documented in the AWS OpsWorks Stacks API Reference. For example, calls to the `CreateLayer`, `DescribeInstances`, and `StartInstance` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understanding OpsWorks Stacks Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateLayer` action.

```
{
```

```
    "Records": [
        {
            "awsRegion": "us-west-2",
            "eventID": "342cd1ec-8214-4a0f-a68f-8e6352feb5af",
            "eventName": "CreateLayer",
            "eventSource": "opsworks.amazonaws.com",
            "eventTime": "2014-05-28T16:05:29Z",
            "eventVersion": "1.01"ed,
            "requestID": "e3952a2b-e681-11e3-aa71-81092480ee2e",
            "requestParameters": {
                "attributes": {},
                "customRecipes": {},
                "name": "2014-05-28 16:05:29 +0000 a073",
                "shortname": "customcf4571d5c0d6",
                "stackId": "a263312e-f937-4949-a91f-f32b6b641b2c",
                "type": "custom"
            },
            "responseElements": null,
            "sourceIPAddress": "198.51.100.0",
            "userAgent": "aws-sdk-ruby/2.0.0 ruby/2.1 x86_64-linux",
            "userIdentity": {
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "accountId": "111122223333",
                "arn": "arn:aws:iam::111122223333:user/A-User-Name",
                "principalId": "AKIAI44QH8DHBEXAMPLE",
                "type": "IAMUser",
                "userName": "A-User-Name"
            }
        },
        {
            "awsRegion": "us-west-2",
            "eventID": "a860d8f8-c1eb-449b-8f55-eafc373b49a4",
            "eventName": "DescribeInstances",
            "eventSource": "opsworks.amazonaws.com",
            "eventTime": "2014-05-28T16:05:31Z",
            "eventVersion": "1.01",
            "requestID": "e4691bfd-e681-11e3-aa71-81092480ee2e",
            "requestParameters": {
                "instanceIds": [
                    "218289c4-0492-473d-a990-3fbe1efa25f6"
                ]
            },
            "responseElements": null,
            "sourceIPAddress": "198.51.100.0",
```

```
            "userAgent": "aws-sdk-ruby/2.0.0 ruby/2.1x86_64-linux",
            "userIdentity": {
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
                "accountId": "111122223333",
                "arn": "arn:aws:iam::111122223333:user/A-User-Name",
                "principalId": "AKIAI44QH8DHBEXAMPLE",
                "type": "IAMUser",
                "userName": "A-User-Name"
            }
        }
    ]
}
```


Using Amazon CloudWatch Logs with OpsWorks Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

To simplify the process of monitoring logs on multiple instances, OpsWorks Stacks supports Amazon CloudWatch Logs. You enable CloudWatch Logs at the layer level in OpsWorks Stacks. CloudWatch Logs integration works with Chef 11.10 and Chef 12 Linux-based stacks. You incur additional charges when you enable CloudWatch Logs, so review Amazon CloudWatch Pricing before you get started.

CloudWatch Logs monitors selected logs for the occurrence of a user-specified pattern. For example, you can monitor logs for the occurrence of a literal term such as `NullReferenceException`, or count the number of such occurrences. After you enable CloudWatch Logs in OpsWorks Stacks, the OpsWorks Stacks agent sends the logs to CloudWatch Logs. For more information about CloudWatch Logs, see Getting Started with CloudWatch Logs.

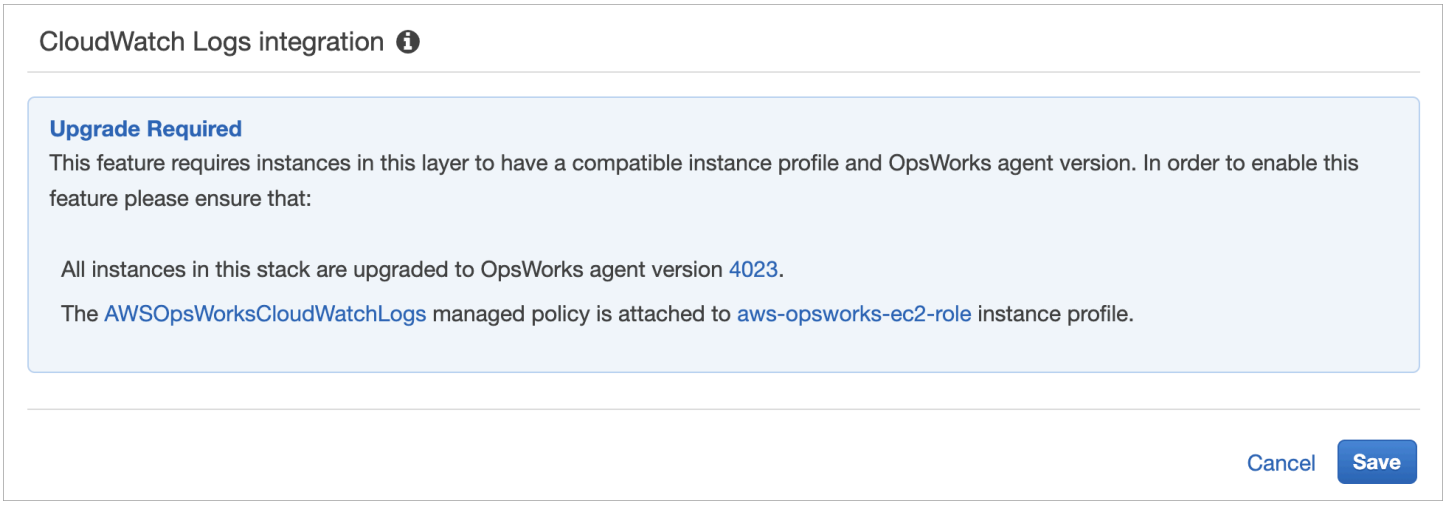
Prerequisites

Before you can enable CloudWatch Logs, your instances must be running version 3444 or later of the OpsWorks Stacks agent in Chef 11.10 stacks, and 4023 or later in Chef 12 stacks. You must also

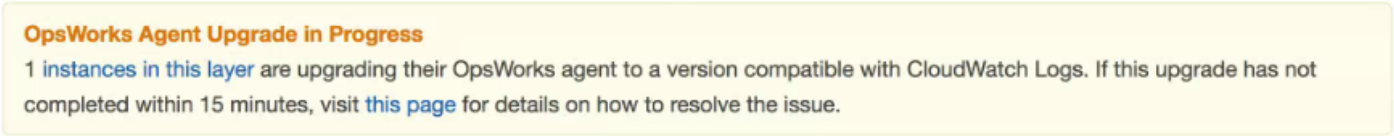
use a compatible instance profile for any instances that you are monitoring by using CloudWatch Logs.

If you are using a custom instance profile (one that OpsWorks Stacks did not provide when you created the stack), OpsWorks Stacks cannot automatically upgrade the instance profile. You must manually attach the **AWSOpsWorksCloudWatchLogs** policy to your profile by using IAM. For information, see Managing IAM policies in the *IAM User Guide*.

If you need to upgrade your agent version or instance profile, OpsWorks Stacks displays a reminder similar to following screen shot when you open the CloudWatch Logs tab on the **Layer** page.



Updating the agent on all instances in a layer can take some time. If you try to enable CloudWatch Logs on a layer before the agent upgrade is complete, you see a message similar to the following.



Enabling CloudWatch Logs

1. After any required agent and instance profile upgrades are complete, you can enable CloudWatch Logs by setting the slider control on the **CloudWatch Logs** tab to **On**.

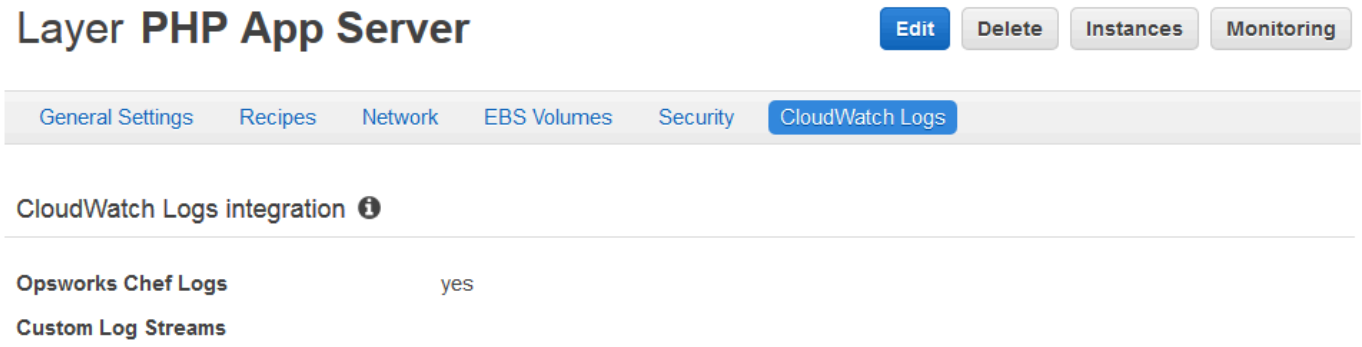


2. To stream command logs, set the **Stream command logs** slider to **On**. This sends logs of Chef activities and user-initiated commands on your layer's instances to CloudWatch Logs.

   The data included in these logs closely matches what you see in the results of a DescribeCommands operation, when you open the target of the log URL. It includes data about `setup`, `configure`, `deploy`, `undeploy`, `start`, `stop`, and recipe run commands.

3. To stream logs of activities that are stored in a custom location on your layer's instances, such as `/var/log/apache/myapp/mylog*`, type the custom location in the **Stream custom logs** string box, and then choose **Add** (**+**).

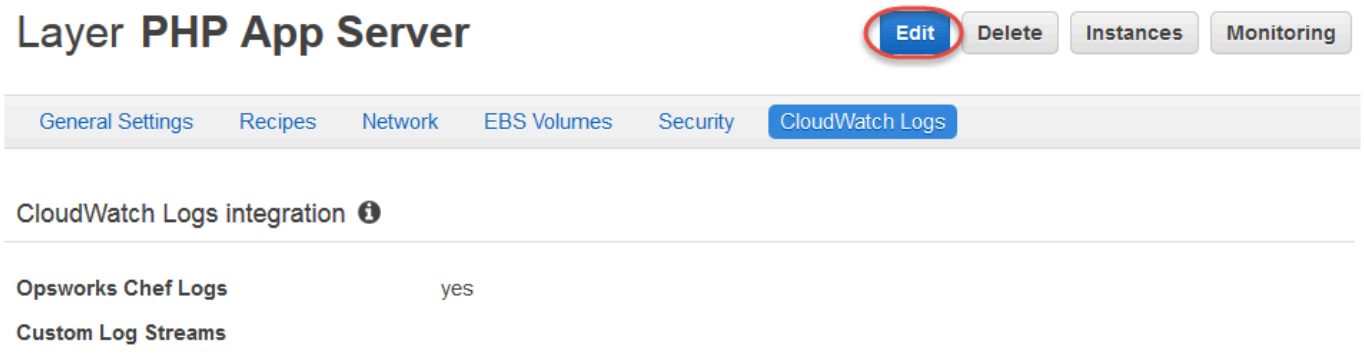
4. Choose **Save**. Within a few minutes, OpsWorks Stacks log streams should be visible in the CloudWatch Logs console.



Turning Off CloudWatch Logs

To turn off CloudWatch Logs, edit your layer settings.

1. On your layer's properties page, choose **Edit**.



2. On the editing page, choose the **CloudWatch Logs** tab.
3. In the **CloudWatch Logs** area, turn off **Stream command logs**. Choose **X** on custom logs to delete them from log streams, if applicable.
4. Choose **Save**.

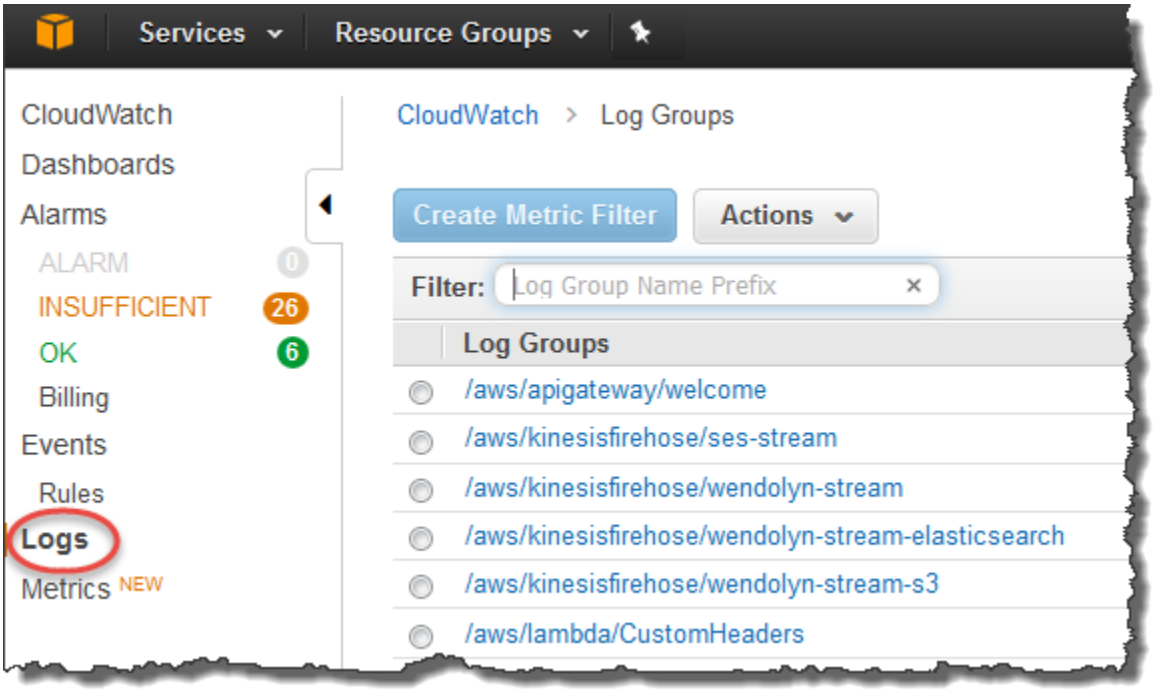
Deleting Streamed Logs from CloudWatch Logs

After you turn off CloudWatch Logs streaming from OpsWorks Stacks, existing logs are still available in the CloudWatch Logs management console. You still incur charges for stored logs, unless you export the logs to Amazon S3 or delete them. For more information about exporting logs to S3, see Exporting Log Data to Amazon S3.

You can delete log streams and log groups in the CloudWatch Logs management console, or by running the `delete-log-stream` and `delete-log-group` AWS CLI commands. For more information about changing log retention periods, see Change Log Data Retention in CloudWatch Logs.

Managing Your Logs in CloudWatch Logs

The logs that you are streaming are managed in the CloudWatch Logs console.



OpsWorks creates default log groups and log streams automatically. Log groups for OpsWorks Stacks data have names that match the following pattern:

stack_name/*layer_name*/*chef_log_name*

Custom logs have names that match the following pattern:

/stack_name/layer_short_name/file_path_name. The path name is made more human-readable by the removal of special characters, such as asterisks (*).

When you've located your logs in CloudWatch Logs, you can organize the logs into groups, search and filter logs by creating metric filters, and create custom alarms.

Configuring Chef 12.2 Windows Layers to Use CloudWatch Logs

CloudWatch Logs automatic integration is not supported for Windows-based instances. The **CloudWatch Logs** tab is not available on layers in Chef 12.2 stacks. To manually enable streaming to CloudWatch Logs for Windows-based instances, do the following.

- Update the instance profile for Windows-based instances so that the CloudWatch Logs agent has appropriate permissions. The **AWSOpsWorksCloudWatchLogs** policy statement shows which permissions are required.

Typically, you do this task only once. You can then use the updated instance profile for all Windows instances in a layer.

- Edit the following JSON configuration file on each instance. This file includes log stream preferences, such as which logs to monitor.

  ```
  %PROGRAMFILES%\Amazon\Ec2ConfigService\Settings
  \AWS.EC2.Windows.CloudWatch.json
  ```

You can automate the preceding two tasks by creating custom recipes to handle the required tasks and assigning them to the Chef 12.2 layer's **Setup** events. Each time you start a new instance on those layers, OpsWorks Stacks automatically runs your recipes after the instance finishes booting, enabling CloudWatch Logs.

To turn off CloudWatch Logs on Windows-based instances, reverse the process. Clear the **Enable CloudWatch Logs integration** check box in the **EC2 Service Properties** dialog box, delete log stream preferences from the `AWS.EC2.Windows.CloudWatch.json` file; and stop running any Chef recipes that are automatically assigning CloudWatch Logs permissions to new instances in Chef 12.2 layers.

Monitoring Stacks using Amazon CloudWatch Events

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can configure rules in Amazon CloudWatch Events to alert you to changes in OpsWorks Stacks resources, and direct CloudWatch Events to take actions based on event contents. For more information about how to get started with CloudWatch Events and set up rules, see Getting Started with CloudWatch Events in the *CloudWatch Events User Guide*.

The following OpsWorks Stacks event types are supported in CloudWatch Events.

Instance state change

Indicates a change in the state of an OpsWorks Stacks instance.

Command state change

Indicates a change occurred in the state of an OpsWorks Stacks command.

Deployment state change

Indicates a change occurred in the state of an OpsWorks Stacks deployment.

Alerts

Indicates an OpsWorks Stacks service error was raised.

For more information about the OpsWorks Stacks event types that are supported by CloudWatch Events, see OpsWorks Stacks Events in the *CloudWatch Events User Guide*.

Security and Permissions

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Each of your users must have appropriate AWS credentials to access your account's AWS resources. The recommended way to provide credentials to users is with AWS Identity and Access Management (IAM). OpsWorks Stacks integrates with IAM to let you control the following:

- How individual users can interact with OpsWorks Stacks.

  For example, you can allow some users to deploy apps to any stack but not modify the stack itself, while allowing other users full access but only to certain stacks, and so on.
- How OpsWorks Stacks can act on your behalf to access stack resources such as Amazon EC2 instances and Amazon S3 buckets.

OpsWorks Stacks provides a service role that grants permissions for these tasks.

- How apps that run on Amazon EC2 instances controlled by OpsWorks Stacks can access other AWS resources, such as data stored on Amazon S3 buckets.

  You can assign an instance profile to a layer's instances that grants permissions to apps running on those instances to access other AWS resources.

- How to manage user-based SSH keys and use SSH or RDP to connect to instances.

  For each stack, administrative users can assign each user a personal SSH key, or authorize users to specify their own key. You can also authorize SSH or RDP access and sudo or administrator privileges on the stack's instances for each user.

Other aspects of security include the following:

- How to manage updating your instances' operating system with the latest security patches.

  For more information, see Managing Security Updates.

- How to configure Amazon EC2 security groups to control network traffic to and from your instances.

  How to specify custom security groups instead of the OpsWorks Stacks default security groups. For more information, see Using Security Groups.

Topics

- Managing OpsWorks Stacks User Permissions
- Allowing OpsWorks Stacks to Act on Your Behalf
- Cross-service confused deputy prevention in OpsWorks Stacks
- Specifying Permissions for Apps Running on EC2 instances
- Managing SSH Access
- Managing Linux Security Updates
- Using Security Groups

Managing OpsWorks Stacks User Permissions

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

As a best practice, restrict OpsWorks Stacks users to a specified set of actions or set of stack resources. You can control OpsWorks Stacks user permissions in two ways: by using the OpsWorks Stacks **Permissions** page, and by applying an appropriate IAM policy.

The OpsWorks **Permissions** page—or the equivalent CLI or API actions—allows you to control user permissions in a multiuser environment on a per-stack basis by assigning each user one of several *permission levels*. Each level grants permissions for a standard set of actions for a particular stack resource. Using the **Permissions** page, you can control the following:

- Who can access each stack.
- Which actions each user is allowed to perform on each stack.

  For example, you can allow some users to only view the stack while others can deploy applications, add instances, and so on.
- Who can manage each stack.

  You can delegate management of each stack to one or more specified users.
- Who has user-level SSH access and sudo privileges (Linux) or RDP access and administrator privileges (Windows) on each stack's Amazon EC2 instances.

  You can grant or remove these permissions separately for each user at any time.

Important

Denying SSH/RDP access does not necessarily prevent a user from logging into instances. If you specify an Amazon EC2 key pair for an instance, any user with the corresponding

private key can log in or use the key to retrieve the Windows administrator password. For more information, see Managing SSH Access.

You can use the IAM console, CLI, or API to add policies to your users that grant explicit permissions for the various OpsWorks Stacks resources and actions.

- Using an IAM policy to specify permissions is more flexible than using the permissions levels.
- You can set up IAM Identities (users, user groups, and roles), which grant permissions to IAM identities, such as users and user groups, or define roles that can be associated with federated users.
- An IAM policy is the only way to grant permissions for certain key OpsWorks Stacks actions.

  For example, you must use IAM to grant permissions for `opsworks:CreateStack` and `opsworks:CloneStack`, which are used to create and clone stacks, respectively.

While it's not explicitly possible to import federated users in the console, a federated user can implicitly create a user profile by choosing **My Settings** at the upper right of the OpsWorks Stacks console, and then choosing **Users**, also at the upper right. On the **Users** page, federated users—whose accounts are created by using the API or CLI, or implicitly through the console—can manage their accounts similarly to non-federated users.

The two approaches are not mutually exclusive and it is sometimes useful to combine them; OpsWorks Stacks then evaluates both sets of permissions. For example, suppose you want to allow users to add or delete instances but not add or delete layers. None of the OpsWorks Stacks permission levels grant that specific set of permissions. However, you can use the **Permissions** page to grant users a **Manage** permission level, which allows them to perform most stack operations, and then apply an IAM policy that denies permissions to add or remove layers. For more information, see Controlling access to AWS resources using policies.

The following is a typical model for managing user permissions. In each case, the reader (you) is assumed to be an administrative user.

1. Use the IAM console to apply AWSOpsWorks_FullAccess policies to one or more administrative users.
2. Create a user for each nonadministrative user with a policy that grants no OpsWorks Stacks permissions.

If a user requires access only to OpsWorks Stacks, you might not need to apply a policy at all. You can instead manage their permissions with the OpsWorks Stacks **Permissions** page.

3. Use the OpsWorks Stacks **Users** page to import the nonadministrative users into OpsWorks Stacks.
4. For each stack, use the stack's **Permissions** page to assign a permission level to each user.
5. As needed, customize users' permission levels by applying an appropriately configured IAM policy.

For more recommendations about managing users, see Best Practices: Managing Permissions.

For more information about IAM best practices, see Security best practices in IAM in the *IAM User Guide*.

Topics

- Managing OpsWorks Stacks Users
- Granting OpsWorks Stacks Users Per-Stack Permissions
- Managing OpsWorks Stacks Permissions by Attaching an IAM Policy
- Example Policies
- OpsWorks Stacks Permissions Levels

Managing OpsWorks Stacks Users

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Before you can import users into OpsWorks Stacks and grant them permissions, you must first have created a user for each individual. To create IAM users, start by signing in to AWS as a user that has been granted the permissions defined in the IAMFullAccess policy. You then use the IAM console to

create IAM users for everyone who needs to access OpsWorks Stacks. You can then import those users into OpsWorks Stacks and grant user permissions as follows:

Regular OpsWorks Stacks Users

Regular users don't require an attached policy. If they do have one, it typically does not include any OpsWorks Stacks permissions. Instead, use the OpsWorks Stacks **Permissions** page to assign one of the following permissions levels to regular users on a stack-by-stack basis.

- **Show** permissions allow users to view the stack, but not perform any operations.
- **Deploy** permissions include the **Show** permissions and also allow users to deploy and update apps.
- **Manage** permissions include the **Deploy** permissions and also allow users to perform stack management operations, such as adding layers or instances, use the **Permissions** page to set user permissions, and enable their own SSH/RDP and sudo/admin privileges.
- **Deny** permissions deny access to the stack.

If these permissions levels are not quite what you want for a particular user, you can customize the user's permissions by applying an IAM policy. For example, you might want to use the OpsWorks Stacks **Permissions** page to assign **Manage** permissions level to a user, which grants them permissions to perform all stack management operations, but not to create or clone stacks. You could then apply a policy that restricts those permissions by denying them permission to add or delete layers or augments those permissions by allowing them to create or clone stacks. For more information, see Managing OpsWorks Stacks Permissions by Attaching an IAM Policy.

OpsWorks Stacks Administrative Users

Administrative users are the account owner or an IAM user with the permissions that are defined by the AWSOpsWorks_FullAccess policy. In addition to the permissions granted to **Manage** users, this policy includes permissions for actions that cannot be granted through the **Permissions** page, such as the following:

- Importing users into OpsWorks Stacks
- Creating and cloning stacks

For the complete policy, see Example Policies. For a detailed list of permissions that can be granted to users only by applying an IAM policy, see OpsWorks Stacks Permissions Levels.

Topics

- Users and Regions
- Creating an OpsWorks Stacks Administrative User
- Creating IAM users for OpsWorks Stacks
- Importing Users into OpsWorks Stacks
- Editing OpsWorks Stacks User Settings

Users and Regions

OpsWorks Stacks users are available within the regional endpoint in which they were created. You can create users in any of the following Regions.

- US East (Ohio) Region
- US East (N. Virginia) Region
- US West (Oregon) Region
- US West (N. California) Region
- Canada (Central) Region (API only; not available in the AWS Management Console
- Asia Pacific (Mumbai) Region
- Asia Pacific (Singapore) Region
- Asia Pacific (Sydney) Region
- Asia Pacific (Tokyo) Region
- Asia Pacific (Seoul) Region
- Europe (Frankfurt) Region
- Europe (Ireland) Region
- Europe (London) Region
- Europe (Paris) Region
- South America (São Paulo) Region

When you import users to OpsWorks Stacks, you import them to one of the regional endpoints; if you want an user to be available in more than one Region, you must import the user to that Region. You can also import OpsWorks Stacks users from one Region to another; if you import a user to a Region that already has a user with the same name, the imported user replaces the existing user. For more information about importing users, see Importing Users.

Creating an OpsWorks Stacks Administrative User

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can create an OpsWorks Stacks administrative user by adding the `AWSOpsWorks_FullAccess` policy to a user, which grants that user OpsWorks Stacks Full Access permissions. For more information about creating an administrative user, see Create an administrative user.

Note

The AWSOpsWorks_FullAccess policy allows users to create and manage OpsWorks Stacks stacks, but users cannot create an IAM service role for the stack; they must use an existing role. The first user to create a stack must have additional IAM permissions, as described in Administrative Permissions. When this user creates the first stack, OpsWorks Stacks creates an IAM service role with the required permissions. Thereafter, any user with `opsworks:CreateStack` permissions can use that role to create additional stacks. For more information, see Allowing OpsWorks Stacks to Act on Your Behalf.

When you create a user, you can add additional customer-managed policies to fine-tune the user's permissions, as needed. For example, you might want an administrative user to be able to create or delete stacks, but not import new users. For more information, see Managing OpsWorks Stacks Permissions by Attaching an IAM Policy.

If you have multiple administrative users, instead of setting permissions separately for each user, you can add the AWSOpsWorks_FullAccess policy to an IAM group and add the users to that group.

For information about creating a group, see Creating IAM user groups. When you create the group, add the **AWSOpsWorks_FullAccess** policy. You can also add the **AdministratorAccess** policy, which includes the **AWSOpsWorks_FullAccess** permissions.

For information about adding permissions to an existing group, see Attaching a policy to an IAM user group.

Creating IAM users for OpsWorks Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Before you can import IAM users into OpsWorks Stacks, you need to create them. You can do this using the IAM console, command line, or API. For full instructions, see Creating an IAM user in your AWS account.

Note that unlike administrative users, you don't need to attach a policy to define permissions. You can set permissions after importing the users into OpsWorks Stacks, as explained in Managing User Permissions.

For more information on creating IAM users and groups, see Getting started with IAM.

Importing Users into OpsWorks Stacks

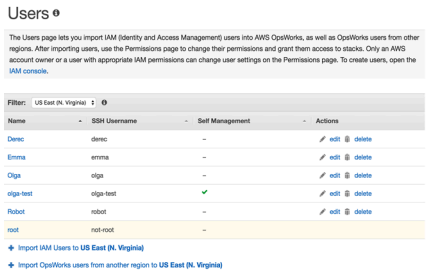
> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Administrative users can import users into OpsWorks Stacks; they can also import OpsWorks Stacks users from one regional endpoint to another. When you import users to OpsWorks Stacks, you import them to one of the OpsWorks Stacks regional endpoints. If you want a user to be available in more than one Region, you must import the user to that Region.

While it's not explicitly possible to import federated users in the console, a federated user can implicitly create a user profile by choosing **My Settings** at the upper right of the OpsWorks Stacks console, and then choosing **Users**, also at the upper right. On the **Users** page, federated users—whose accounts are created by using the API or CLI, or implicitly through the console—can manage their accounts similarly to non-federated users.

To import users into OpsWorks Stacks

1. Sign in to OpsWorks Stacks as an administrative user or as the account owner.
2. Choose **Users** at the upper right to open the **Users** page.



3. Choose **Import IAM Users to <*region name*>** to display the users that are available, but that have not yet been imported.



4. Fill the **Select all** check box, or select one or more individual users. When you are finished, choose **Import to OpsWorks**.

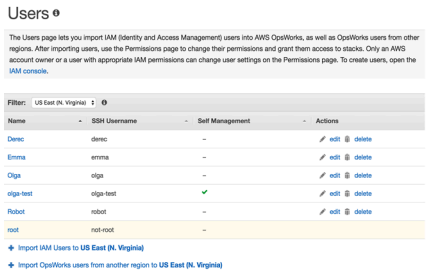
> **Note**
>
> After you have imported a user into OpsWorks Stacks, if you use the IAM console or API to delete the user from your account, the user does not automatically lose SSH access that you have granted through OpsWorks Stacks. You must also delete the user from OpsWorks Stacks by opening the **Users** page, and choosing **delete** in the user's **Actions** column.

To import OpsWorks Stacks users from one Region to another

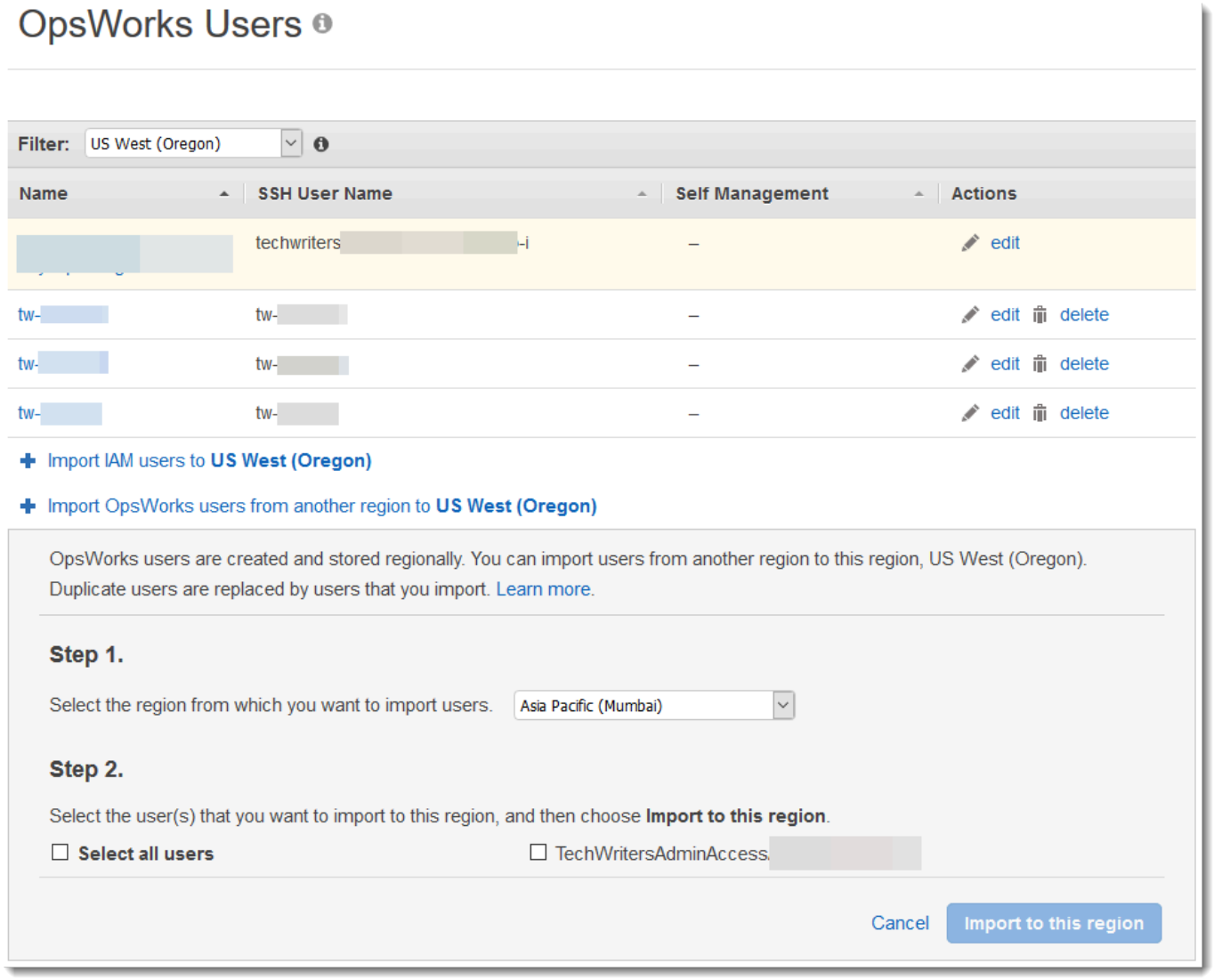
OpsWorks Stacks users are available within the regional endpoint in which they were created. You can create users in the Regions shown in Users and Regions.

You can import OpsWorks Stacks users from one Region to the Region to which your **Users** list is currently filtered. If you import a user to a Region that already has a user with the same name, the imported user replaces the existing user.

1. Sign in to OpsWorks Stacks as an administrative user or as the account owner.
2. Choose **Users** at the upper right to open the **Users** page. If you have OpsWorks Stacks users in more than one Region, use the **Filter** control to filter for the Region to which you want to import users.



3. Choose **Import OpsWorks Stacks users from another Region to <*current region*>**.



4. Select the Region from which you want to import OpsWorks Stacks users.
5. Select one or more users to import, or select all users, and then choose **Import to this region**. Wait for OpsWorks Stacks to display the imported users in the **Users** list.

Unix IDs and Users Created Outside OpsWorks Stacks

OpsWorks assigns users on OpsWorks Stacks instances Unix ID (UID) values between 2000 and 4000. Because OpsWorks reserves the 2000-4000 range of UIDs, users that you create outside of OpsWorks (by using cookbook recipes, or by importing users into OpsWorks from IAM, for example) can have UIDs that are overwritten by OpsWorks Stacks for another user. This can result in users that you have created outside of OpsWorks Stacks not showing up in data bag search results, or being excluded from the OpsWorks Stacks built-in `sync_remote_users` operation.

External processes can also create users with UIDs that OpsWorks Stacks can overwrite. Some operating system packages, for example, can create a user as part of post-installation processes. When you or a software process creates a user on a Linux-based operating system without explicitly specifying a UID—which is the default—the UID assigned by OpsWorks Stacks is *<highest existing OpsWorks UID>* + 1.

As a best practice, create OpsWorks Stacks users and manage their access in the OpsWorks Stacks console, AWS CLI, or by using an AWS SDK. If you do create users on OpsWorks Stacks instances outside of OpsWorks, use *UnixID* values greater than 4000.

Editing OpsWorks Stacks User Settings

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

After you have imported users, you can edit their settings, as follows:

To edit user settings

1. On the **Users** page, choose **edit** in the user's **Actions** column.
2. You can specify the following settings.

   Self Management

   Select **Yes** to allow the user to use the MySettings page to specify his or her personal SSH key.

   > **Note**
   >
   > You can also enable self-management by adding IAM policy to the IAM identity that grants permissions for the DescribeMyUserProfile and UpdateMyUserProfile actions.

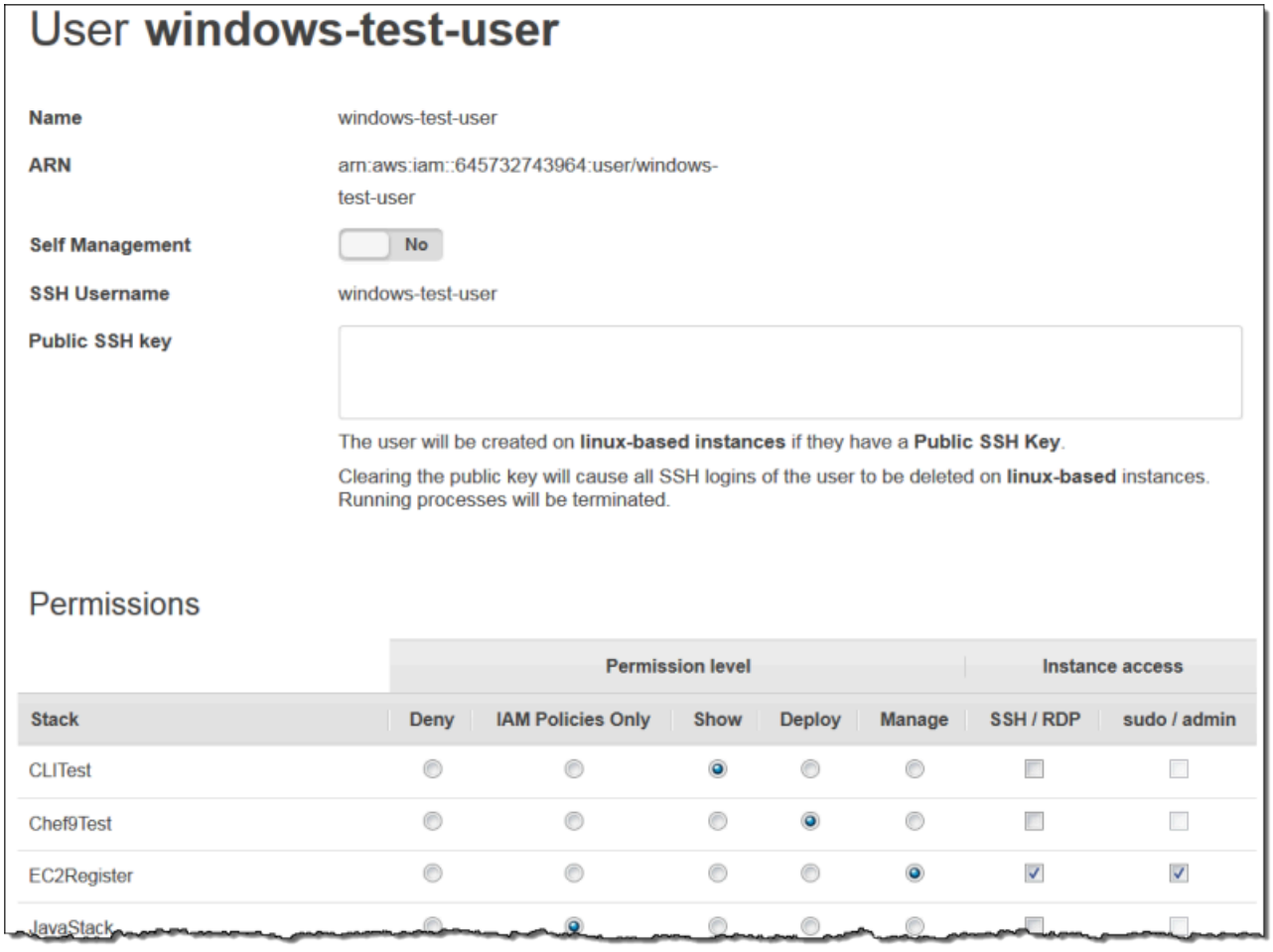
Public SSH key

(Optional) Enter a public SSH key for the user. This key will appear on the user's **My Settings** page. If you enable self-management, the user can edit **My Settings** and specify his or her own key. For more information, see Registering a User's Public SSH Key.

OpsWorks Stacks installs this key on all Linux instances; users can use the associated private key to log in. For more information, see Logging In with SSH. You cannot use this key with Windows stacks.

Permissions

(Optional) Set the user's permissions levels for each stack in one place instead of setting them separately by using each stack's **Permissions** page. For more information on permissions levels, see Granting Per-Stack Permissions.



Granting OpsWorks Stacks Users Per-Stack Permissions

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The simplest way to manage OpsWorks Stacks user permissions is by using a stack's **Permissions** page. Each stack has its own page, which grants permissions for that stack.

You must be signed in as an administrative user or **Manage** user to modify any of the permissions settings. The list shows only those users that have been imported into OpsWorks Stacks. For information on how to create and import users, see Managing Users.

The default permission level is IAM Policies Only, which grants users only those permissions that are in their IAM policy.

- When you import a user from IAM or from another Region, the user is added to the list for all existing stacks with an **IAM Policies Only** permission level.
- By default, a user whom you have just imported from another Region has no access to stacks in the destination Region. If you import users from another Region, to let them manage stacks in the destination Region, they must be assigned permissions to those stacks after you import the users.
- When you create a new stack, all current users are added to the list with **IAM Policies Only** permission levels.

Topics

- Setting a User's Permissions
- Viewing your Permissions
- Using IAM Condition Keys to Verify Temporary Credentials

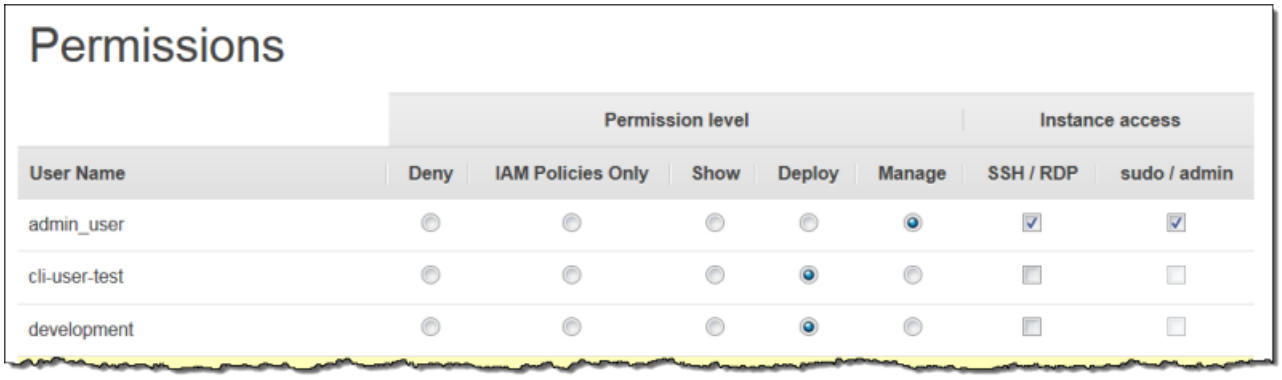
Setting a User's Permissions

To set a user's permissions

1. In the navigation pane, choose **Permissions**.
2. On the **Permissions** page, choose **Edit**.
3. Change the **Permission level** and **Instance access** settings:

   - Use the **Permissions level** settings to assign one of the standard permission levels to each user, which determine whether the user can access this stack and what actions the user can perform. If a user has an IAM policy, OpsWorks Stacks evaluates both sets of permissions. For an example see Example Policies.
   - The **Instance access SSH/RDP** setting specifies whether the user has SSH (Linux) or RDP (Windows) access to the stack's instances.

     If you authorize **SSH/RDP** access, you can optionally select **sudo/admin**, which grants the user sudo (Linux) or administrative (Windows) privileges on the stack's instances.



You can assign each user to one of the following permissions levels. For a list of the actions that are allowed by each level, see OpsWorks Stacks Permissions Levels.

Deny

The user cannot perform any OpsWorks Stacks actions on the stack, even if they have an IAM policy that grants OpsWorks Stacks full access permissions. You might use this, for example, to deny some users access to stacks for unreleased products.

IAM Policies Only

The default level, which is assigned to all newly imported users, and to all users for newly created stacks. The user's permissions are determined by their IAM policy. If a user has no IAM policy, or their policy has no explicit OpsWorks Stacks permissions, they cannot access the stack. Administrative users are typically assigned this level because their IAM policies already grant full access permissions.

Show

The user can view a stack, but not perform any operations. For example, managers might want to monitor an account's stacks, but would not need to deploy apps or modify the stack in any way.

Deploy

Includes the **Show** permissions and also allows the user to deploy apps. For example, an app developer might need to deploy updates to the stack's instances but not add layers or instances to the stack.

Manage

Includes the **Deploy** permissions and also allows the user to perform a variety of stack management operations, including:

- Adding or deleting layers and instances.
- Using the stack's **Permissions** page to assign permissions levels to users.
- Registering or deregistering resources.

For example, each stack can have a designated manager who is responsible for ensuring that the stack has an appropriate number and type of instances, handling package and operating system updates, and so on.

> **Note**
>
> The Manage level does not let users create or clone stacks. Those permissions must be granted by an IAM policy. For an example, see Manage Permissions.

If the user also has an IAM policy, OpsWorks Stacks evaluates both sets of permissions. This allows you to assign a permission level to a user and then apply a policy to restrict or augment the level's

allowed actions. For example, you could apply a policy that allows a **Manage** user to create or clone stacks, or denies that user the ability to register or deregister resources. For some examples of such policies, see Example Policies.

> **Note**
>
> If the user's policy allows additional actions, the result can appear to override the **Permissions** page settings. For example, if a user has a policy that allows the CreateLayer action but you use the **Permissions** page to specify **Deploy** permissions, the user is still allowed to create layers. The exception to this rule is the **Deny** option, which denies stack access even to users with AWSOpsWorks_FullAccess policies. For more information, see Controlling access to AWS resources using policies.

Viewing your Permissions

If self-management is enabled, users can see a summary of their permission levels for every stack by choosing **My Settings**, on the upper right. Users can also access **My Settings** if their policy grants permissions for the DescribeMyUserProfile and UpdateMyUserProfile actions.

Using IAM Condition Keys to Verify Temporary Credentials

OpsWorks Stacks has a built-in authorization layer that supports additional authorization cases (such as the simplified management of read-only or read-write access to stacks for individual users). This authorization layer relies on the usage of temporary credentials. Because of this, you cannot use an `aws:TokenIssueTime` condition to verify that users are using long-term credentials, or block actions from users who are using temporary credentials, as described in IAM JSON policy elements reference in the IAM documentation.

Managing OpsWorks Stacks Permissions by Attaching an IAM Policy

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can specify a user's OpsWorks Stacks permissions by attaching an IAM policy. An attached policy is required for some permissions:

- Administrative user permissions, such as importing users.
- Permissions for some actions, such as creating or cloning a stack.

For a complete list of actions that require an attached policy, see OpsWorks Stacks Permissions Levels.

You can also use a policy to customize permission levels that were granted through the **Permissions** page. This section provides a brief summary of how to apply an IAM policy to a user to specify OpsWorks Stacks permissions. For more information, see Access management for AWS resources.

An IAM policy is a JSON object that contains one or more *statements*. Each statement element has a list of permissions, which have three basic elements of their own:

Action

The actions that the permission affects. You specify OpsWorks Stacks actions as `opsworks:`*`action`*. An `Action` can be set to a specific action such as `opsworks:CreateStack`, which specifies whether the user is allowed to call `CreateStack`. You can also use wildcards to specify groups of actions. For example, `opsworks:Create*` specifies all creation actions. For a complete list of OpsWorks Stacks actions, see the OpsWorks Stacks API Reference.

Effect

Whether the specified actions are allowed or denied.

Resource

The AWS resources that the permission affects. OpsWorks Stacks has one resource type, the stack. To specify permissions for a particular stack resource, set `Resource` to the stack's ARN, which has the following format: `arn:aws:opsworks:`*`region`*`:`*`account_id`*`:stack/`*`stack_id`*`/`.

You can also use wildcards. For example, setting `Resource` to `*` grants permissions for every resource.

For example, the following policy denies the user the ability to stop instances on the stack whose ID is `2860-2f18b4cb-4de5-4429-a149-ff7da9f0d8ee`.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "opsworks:StopInstance",
      "Effect": "Deny",
      "Resource": "arn:aws:opsworks:*:*:stack/2f18b4cb-4de5-4429-a149-
ff7da9f0d8ee/"
    }
  ]
}
```

For information about adding permissions to an IAM user, see https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users_change-permissions.html#users_change_permissions-add-console.

For more information about how to create or modify IAM policies, see Policies and permissions in IAM. For some examples of OpsWorks Stacks policies, see Example Policies.

Example Policies

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section describes example IAM policies that can be applied to OpsWorks Stacks users.

- Administrative Permissions describes policies used to grant permissions to administrative users.
- Manage Permissions and Deploy Permissions show examples of policies that can be applied to a user to augment or restrict the Manage and Deploy permissions levels.

OpsWorks Stacks determines the user's permissions by evaluating the permissions granted by IAM policies as well as the permissions granted by the **Permissions** page. For more information, see Controlling access to AWS resources using policies. For more information on the **Permissions** page permissions, see OpsWorks Stacks Permissions Levels.

Administrative Permissions

Use the IAM console, https://console.aws.amazon.com/iam/, to access the AWSOpsWorks_FullAccess policy, Attach this policy to a user to grant them permissions to perform all OpsWorks Stacks actions. The IAM permissions are required, among other things, to allow an administrative user to import users.

You must create an IAM roles that allows OpsWorks Stacks to act on your behalf to access other AWS resources, such as Amazon EC2 instances. You typically handle this task by having an administrative user create the first stack, and letting OpsWorks Stacks create the role for you. You can then use that role for all subsequent stacks. For more information, see Allowing OpsWorks Stacks to Act on Your Behalf.

The administrative user who creates the first stack must have permissions for some IAM actions that are not included in the AWSOpsWorks_FullAccess policy. Add the following permissions to the `Actions` section of the policy. For proper JSON syntax, be sure to add commas between actions and remove the trailing comma at the end of the list of actions.

```
"iam:PutRolePolicy",
"iam:AddRoleToInstanceProfile",
"iam:CreateInstanceProfile",
"iam:CreateRole"
```

Manage Permissions

The **Manage** permissions level allows a user to perform a variety of stack management actions, including adding or deleting layers. This topic describes several policies that you can use to **Manage** users to augment or restrict the standard permissions.

Deny a **Manage** user the ability to add or delete layers

You can restrict the **Manage** permissions level to allow a user perform all **Manage** actions except adding or deleting layers by using the following IAM policy. Replace *`region`*, *`account_id`*, and *`stack_id`* with values appropriate to your configuration.

Allow a **Manage** user to create or clone stacks

The **Manage** permissions level doesn't allow users to create or clone stacks. You can change the **Manage** permissions to allow a user to create or clone stacks by applying the following IAM policy. Replace *region* and *account_id* with values appropriate to your configuration.

Deny a Manage user the ability to register or deregister resources

The **Manage** permissions level allows the user to register and deregister Amazon EBS and Elastic IP address resources with the stack. You can restrict the **Manage** permissions to allow the user to perform all **Manage** actions except registering resources by applying the following policy.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "opsworks:RegisterVolume",
        "opsworks:RegisterElasticIp"
      ],
      "Resource": "*"
    }
  ]
}
```

Allow a **Manage** user to import users

The **Manage** permissions level doesn't allow users to import users into OpsWorks Stacks. You can augment the **Manage** permissions to allow a user to import and delete users by applying the following IAM policy. Replace *region* and *account_id* with values appropriate to your configuration.

Deploy Permissions

The **Deploy** permissions level doesn't allow users to create or delete apps. You can augment the **Deploy** permissions to allow a user to create and delete apps by applying the following IAM policy. Replace *region*, *account_id*, and *stack_id* with values appropriate to your configuration.

OpsWorks Stacks Permissions Levels

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section lists the actions that are allowed by the **Show**, **Deploy**, and **Manage** permissions levels on the OpsWorks Stacks **Permissions** page. It also includes a list of actions that you can grant permissions only by applying an IAM policy to the user.

Show

The **Show** level allows `Describe`*`XYZ`* commands, with the following exceptions:

```
DescribePermissions
DescribeUserProfiles
DescribeMyUserProfile
DescribeStackProvisioningParameters
```

If an administrative user has enabled self-management for the user, **Show** users can also use `DescribeMyUserProfile` and `UpdateMyUserProfile`. For more information on self management, see Editing User Settings.

Deploy

The following actions are allowed by the **Deploy** level, in addition to the actions allowed by the **Show** level.

```
CreateDeployment
UpdateApp
```

Manage

The following actions are allowed by the **Manage** level, in addition to the actions allowed by the **Deploy** and **Show** levels.

```
AssignInstance
AssignVolume
AssociateElasticIp
AttachElasticLoadBalancer
CreateApp
CreateInstance
CreateLayer
DeleteApp
DeleteInstance
DeleteLayer
DeleteStack
DeregisterElasticIp
DeregisterInstance
DeregisterRdsDbInstance
DeregisterVolume
DescribePermissions
DetachElasticLoadBalancer
DisassociateElasticIp
GrantAccess
GetHostnameSuggestion
RebootInstance
RegisterElasticIp
RegisterInstance
RegisterRdsDbInstance
RegisterVolume
SetLoadBasedAutoScaling
SetPermission
SetTimeBasedAutoScaling
StartInstance
StartStack
StopInstance
StopStack
UnassignVolume
UpdateElasticIp
UpdateInstance
UpdateLayer
UpdateRdsDbInstance
UpdateStack
UpdateVolume
```

Permissions That Require an IAM Policy

You must grant permissions for the following actions by applying an appropriate IAM policy to the user. For some examples, see Example Policies.

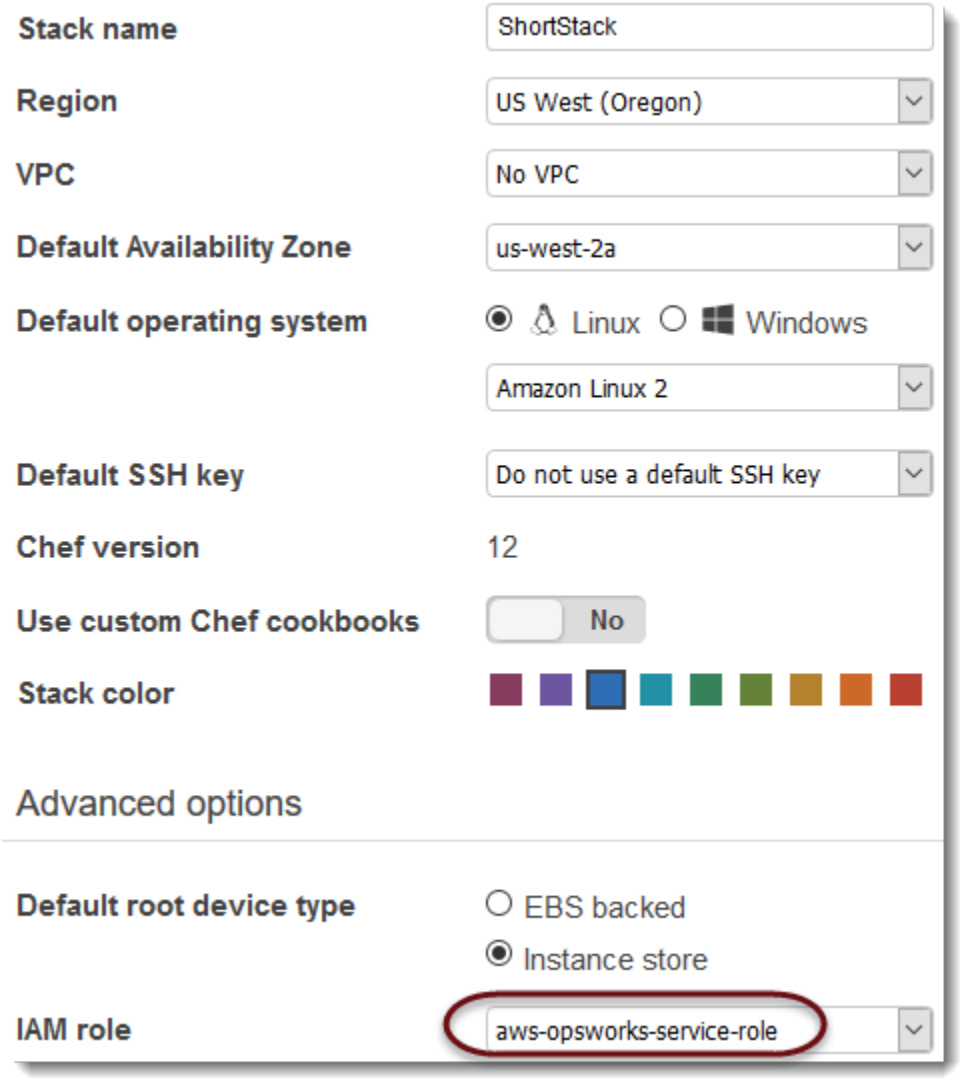
```
CloneStack
CreateStack
CreateUserProfile
DeleteUserProfile
DescribeUserProfiles
UpdateUserProfile
```

Allowing OpsWorks Stacks to Act on Your Behalf

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks needs to interact with a variety of AWS services on your behalf. For example, OpsWorks Stacks interacts with Amazon EC2 to create instances and with Amazon CloudWatch to get monitoring statistics. When you create a stack, you specify an IAM role, usually called a service role, that grants OpsWorks Stacks the appropriate permissions.



When you specify a new stack's service role, you can do one of the following:

- Specify a standard service role that you created earlier.

  You can usually create a standard service role when you create your first stack, and then use that role for all subsequent stacks.
- Specify a custom service role that you created by using the IAM console or API.

  This approach is useful if you want to grant OpsWorks Stacks more limited permissions than the standard service role.

> **Note**
>
> To create your first stack, you must have the permissions defined in the IAM **AdministratorAccess** policy template. These permissions allow OpsWorks Stacks to create a new IAM service role and allow you to import users, as described earlier. For all subsequent stacks, users can select the service role created for the first stack; they don't require full administrative permissions to create a stack.

The standard service role grants the following permissions:

- Perform all Amazon EC2 actions (`ec2:*`).
- Get CloudWatch statistics (`cloudwatch:GetMetricStatistics`).
- Use Elastic Load Balancing to distribute traffic to servers (`elasticloadbalancing:*`).
- Use an Amazon RDS instance as a database server (`rds:*`).
- Use IAM roles (`iam:PassRole`) to provide secure communication between OpsWorks Stacks and your Amazon EC2 instances.

If you create a custom service role, you must ensure that it grants all the permissions that OpsWorks Stacks needs to manage your stack. The following JSON sample is the policy statement for the standard service role; a custom service role should include at least the following permissions in its policy statement.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ec2:*",
                "iam:PassRole",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:DescribeAlarms",
                "ecs:*",
                "elasticloadbalancing:*",
                "rds:*"
            ],
```

```
            "Effect": "Allow",
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "ec2.amazonaws.com"
                }
            }
        }
    ]
}
```

A service role also has a trust relationship. Service roles created by OpsWorks Stacks have the following trust relationship.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "StsAssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": "opsworks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

The service role must have this trust relationship for OpsWorks Stacks to act on your behalf. If you use the default service role, do not modify the trust relationship. If you are creating a custom service role, specify the trust relationship by doing one of the following:

- If you are using the **Create role** wizard in the IAM console, in **Choose a use case**, choose **Opsworks**. This role has the appropriate trust relationship, but no policy is implicitly attached.

To grant OpsWorks Stacks permissions to act on your behalf, create a customer-managed policy that contains the following, and attach it to the new role.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:DescribeAlarms",
        "cloudwatch:GetMetricStatistics",
        "ec2:*",
        "ecs:*",
        "elasticloadbalancing:*",
        "iam:GetRolePolicy",
        "iam:ListInstanceProfiles",
        "iam:ListRoles",
        "iam:ListUsers",
        "rds:*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "ec2.amazonaws.com"
        }
      }
    }
  ]
}
```

- If you are using a AWS CloudFormation template, you can add something like the following to your template's **Resources** section.

```
"Resources": {
  "OpsWorksServiceRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
            "Statement": [ {
              "Effect": "Allow",
              "Principal": {
                  "Service": [ "opsworks.amazonaws.com" ]
              },
              "Action": [ "sts:AssumeRole" ]
            } ]
        },
        "Path": "/",
        "Policies": [ {
            "PolicyName": "opsworks-service",
            "PolicyDocument": {
              ...
            } ]
        }
      },
   }
}
```

Cross-service confused deputy prevention in OpsWorks Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service

can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in stack access policies to limit the permissions that AWS OpsWorks Stacks gives another service to stacks. If the `aws:SourceArn` value does not contain the account ID, such as an Amazon S3 bucket ARN, you must use both global condition context keys to limit permissions. If you use both global condition context keys and the `aws:SourceArn` value contains the account ID, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement. Use `aws:SourceArn` if you want only one stack to be associated with the cross-service access. Use `aws:SourceAccount` if you want to allow any stack in that account to be associated with the cross-service use.

The value of `aws:SourceArn` must be the ARN of an OpsWorks stack.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the OpsWorks Stacks stack. If you don't know the full ARN, or if you are specifying multiple stack ARNs, use the `aws:SourceArn` global context condition key with wildcards (*) for the unknown portions of the ARN. For example, `arn:aws:`*`servicename`*`:*:`*`123456789012`*`:*`.

The following section shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in OpsWorks Stacks to prevent the confused deputy problem.

Prevent confused deputy exploits in OpsWorks Stacks

This section describes how you can help prevent confused deputy exploits in OpsWorks Stacks, and includes examples of permissions policies that you can attach to the IAM role you are using to access OpsWorks Stacks. As a security best practice, we recommend adding the `aws:SourceArn` and `aws:SourceAccount` condition keys to the trust relationships your IAM role has with other services. The trust relationships allow OpsWorks Stacks to assume a role to perform actions in other services that are required to create or manage your OpsWorks Stacks stacks.

To edit trust relationships to add `aws:SourceArn` and `aws:SourceAccount` condition keys

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Roles**.

3. In the **Search** box, search for the role that you use for access to OpsWorks Stacks. The AWS managed role is `aws-opsworks-service-role`.
4. On the **Summary** page for the role, choose the **Trust relationships** tab.
5. On the **Trust relationships** tab, choose **Edit trust policy**.
6. On the **Edit trust policy** page, add at least one of the `aws:SourceArn` or `aws:SourceAccount` condition keys to the policy. Use `aws:SourceArn` to restrict the trust relationship between cross services (such as Amazon EC2) and OpsWorks Stacks to specific OpsWorks Stacks stacks, which is more restrictive. Add `aws:SourceAccount` to restrict the trust relationship between cross services and OpsWorks Stacks to stacks in a specific account, which is less restrictive. The following is an example. Note that if you use both condition keys, the account IDs must be the same.
7. When you are finished adding condition keys, choose **Update policy**.

The following are additional examples of roles that limit access to stacks by using `aws:SourceArn` and `aws:SourceAccount`.

Topics

- Example: Accessing stacks in a specific region
- Example: Adding more than one stack ARN to aws:SourceArn

Example: Accessing stacks in a specific region

The following role trust relationship statement accesses any OpsWorks Stacks stacks in the US East (Ohio) Region (`us-east-2`). Note that the region is specified in the ARN value of `aws:SourceArn`, but the stack ID value is a wildcard (*).

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "opsworks.amazonaws.com"
      },
```

```
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:opsworks:us-east-2:123456789012:stack/*"
        }
      }
    }
  ]
}
```

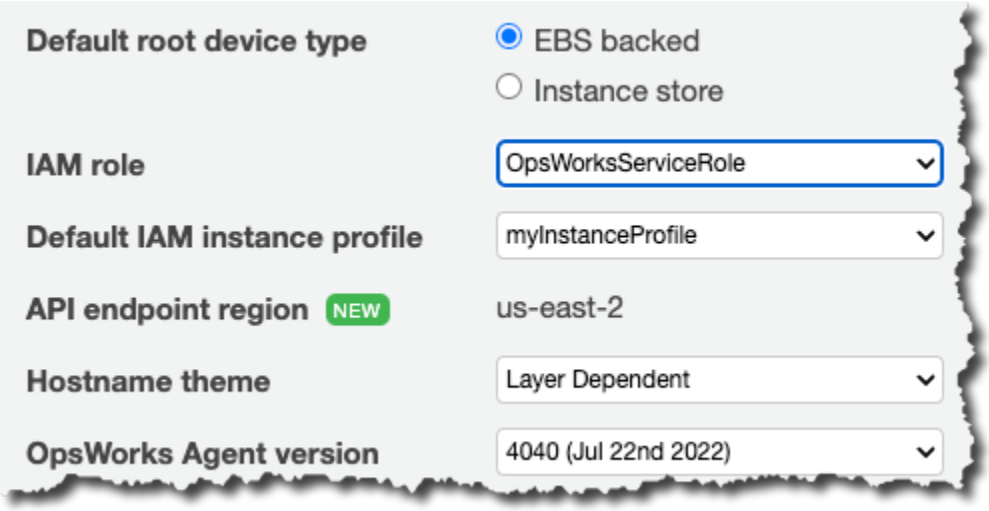
Example: Adding more than one stack ARN to `aws:SourceArn`

The following example limits access to an array of two OpsWorks Stacks stacks in account ID 123456789012.

Specifying Permissions for Apps Running on EC2 instances

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

If the applications running on your stack's Amazon EC2 instances need to access other AWS resources, such as Amazon S3 buckets, they must have appropriate permissions. To confer those permissions, you use an instance profile. You can specify an instance profile for each instance when you create an OpsWorks Stacks stack.



You can also specify a profile for a layer's instances by editing the layer configuration.

The instance profile specifies an IAM role. Applications running on the instance can assume that role to access AWS resources, subject to the permissions that are granted by the role's policy. For more information about how an application assumes a role, see Assuming the Role Using an API Call.

You can create an instance profile in any of the following ways:

- Use the IAM console or API to create a profile.

  For more information, see Roles (Delegation and Federation).
- Use an AWS CloudFormation template to create a profile.

  For some examples of how to include IAM resources in a template, see Identity and Access Management (IAM) Template Snippets.

An instance profile must have a trust relationship and an attached policy that grants permissions to access AWS resources.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
```

```
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

The instance profile must have this trust relationship for OpsWorks Stacks to act on your behalf. If you use the default service role, do not modify the trust relationship. If you are creating a custom service role, specify the trust relation ship as follows:

- If you are using the **Create Role** wizard in the IAM console, specify the **Amazon EC2** role type under **AWS Service Roles** on the wizard's second page.
- If you are using a AWS CloudFormation template, you can add something like the following to your template's **Resources** section.

```
"Resources": {
      "OpsWorksEC2Role": {
         "Type": "AWS::IAM::Role",
         "Properties": {
            "AssumeRolePolicyDocument": {
               "Statement": [ {
                  "Effect": "Allow",
                  "Principal": {
                     "Service": [ "ec2.amazonaws.com" ]
                  },
                  "Action": [ "sts:AssumeRole" ]
               } ]
            },
            "Path": "/"
         }
      },
      "RootInstanceProfile": {
         "Type": "AWS::IAM::InstanceProfile",
         "Properties": {
            "Path": "/",
            "Roles": [ {
               "Ref": "OpsWorksEC2Role"
            }
```

```
            ]
        }
    }
}
```

When you create your instance profile, you can attach an appropriate policy to the profile's role at that time. After you have created the stack, you must use the IAM console or API to attach an appropriate policy to the profile's role. For example, the following policy grants full access to all objects in the Amazon S3 bucket named amzn-s3-demo-bucket. Replace *region* and amzn-s3-demo-bucket with values appropriate to your configuration.

For an example of how to create and use an instance profile, see Using an Amazon S3 Bucket.

If your application uses an instance profile to call the OpsWorks Stacks API from an EC2 instance, the policy must allow the `iam:PassRole` action in addition to the appropriate actions for OpsWorks Stacks and other AWS services. The `iam:PassRole` permission allows OpsWorks Stacks to assume the service role on your behalf. For more information about the OpsWorks Stacks API, see AWS OpsWorks API Reference.

The following is an example of an IAM policy that allows you to call any OpsWorks Stacks action from an EC2 instance, as well as any Amazon EC2 or Amazon S3 action.

Note

If you do not allow `iam:PassRole`, any attempt to call an OpsWorks Stacks action fails with an error like the following:

```
User: arn:aws:sts::123456789012:federated-user/Bob is not authorized
to perform: iam:PassRole on resource:
arn:aws:sts::123456789012:role/OpsWorksStackIamRole
```

For more information about using roles on an EC2 instance for permissions, see Granting Applications that Run on Amazon EC2 Instances Access to AWS Resources in the *AWS Identity and Access Management User Guide*.

Managing SSH Access

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks supports SSH keys for both Linux and Windows stacks.

- For Linux instances, you can use SSH to log in to an instance, for example, to run agent CLI commands.

  For more information, see Logging In with SSH.
- For Windows instances, you can use an SSH key to obtain the instance's Administrator password, which you can then use to log in with RDP.

  For more information, see Logging In with RDP.

Authentication is based on an SSH key pair, which consists of a public key and a private key:

- You install the public key on the instance.

  The location depends on the particular operating system, but OpsWorks Stacks handles the details for you.
- You store the private key locally and provide it to an SSH client, such as `ssh.exe`, to access the instance.

  The SSH client uses the private key to connect to the instance.

To provide SSH access to a stack's users, you need a way to create SSH key pairs, install public keys on the stack's instances, and securely manage the private keys.

Amazon EC2 provides a simple way to install a public SSH key on an instance. You can use the Amazon EC2 console or API to create one or more key pairs for each AWS region that you plan to

use. Amazon EC2 stores the public keys on AWS and you store the private keys locally. When you launch an instance, you specify one of the region's key pairs and Amazon EC2 automatically installs it on the instance. You then use the corresponding private key to log in to the instance. For more information, see Amazon EC2 Key Pairs.

With OpsWorks Stacks, you can specify one of the region's Amazon EC2 key pairs when you create a stack, and optionally override it with a different key pair when you create each instance. When OpsWorks Stacks launches the corresponding Amazon EC2 instance, it specifies the key pair and Amazon EC2 installs the public key on the instance. You can then use the private key to log in or retrieve an Administrator password, just as you would with a standard Amazon EC2 instance. For more information, see Installing an Amazon EC2 Key.

Using an Amazon EC2 key pair is convenient, but has two significant limitations:

- An Amazon EC2 key pair is tied to a particular AWS region.

  If you work in multiple regions, you must manage multiple key pairs.
- You can install only one Amazon EC2 key pair on an instance.

  If you want to allow multiple users to log in, they must all have a copy of the private key, which is not a recommended security practice.

For Linux stacks, OpsWorks Stacks provides a simpler and more flexible way to manage SSH key pairs.

- Each user registers a personal key pair.

  They store the private key locally and register the public key with OpsWorks Stacks, as described in Registering a User's Public SSH Key.
- When you set user permissions for a stack, you specify which users should have SSH access to the stack's instances.

  OpsWorks Stacks automatically creates a system user on the stack's instances for each authorized user and installs their public key. The user can then use the corresponding private key to log in, as described in Logging In with SSH.

Using personal SSH keys has the following advantages.

- There's no need to manually configure keys on the instances; OpsWorks Stacks automatically installs the appropriate public keys on every instance.
- OpsWorks Stacks installs only authorized users' personal public keys.

  Unauthorized users cannot use their personal private key to gain access to instances. With Amazon EC2 key pairs, any user with the corresponding private key can log in, with or without authorized SSH access.
- If a user no longer needs SSH access, you can use the **Permissions** page to revoke the user's SSH/RDP permissions.

  OpsWorks Stacks immediately uninstalls the public key from the stack's instances.
- You can use the same key for any AWS region.

  Users have to manage only one private key.
- There is no need to share private keys.

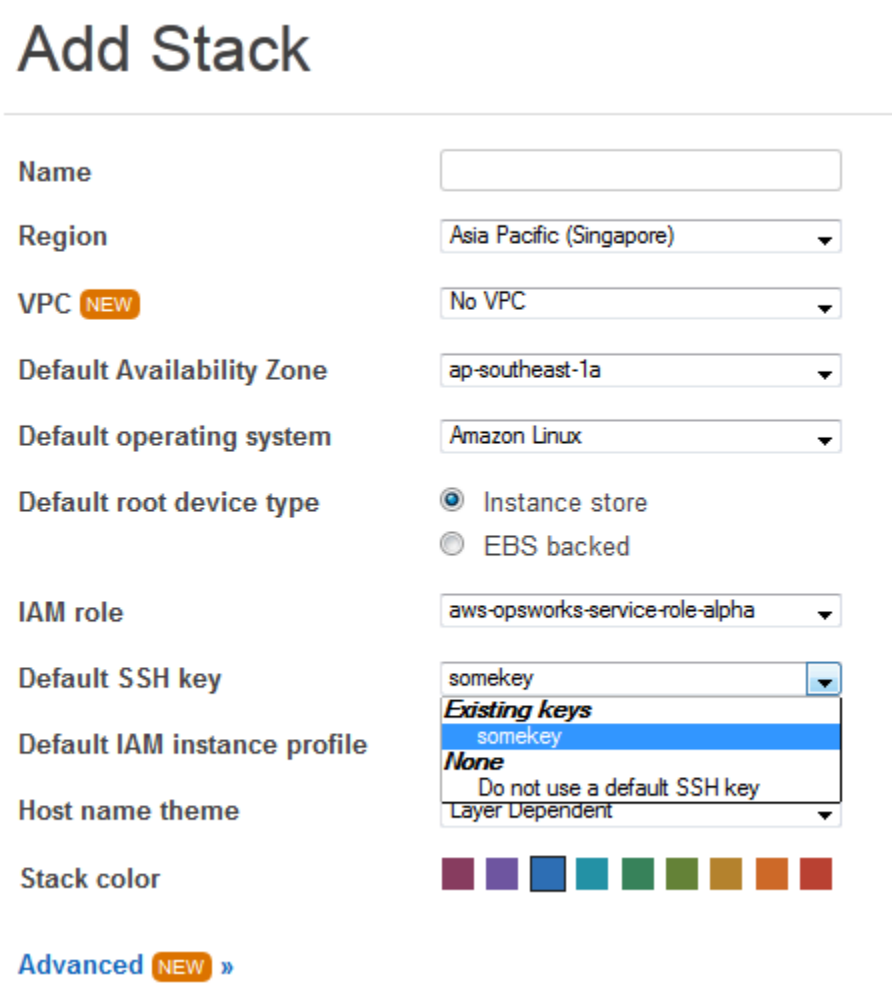
  Each user has his or her own private key.
- It's easy to rotate keys.

  You or the user updates the public key in **My Settings** and OpsWorks Stacks automatically updates the instances.

Installing an Amazon EC2 Key

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

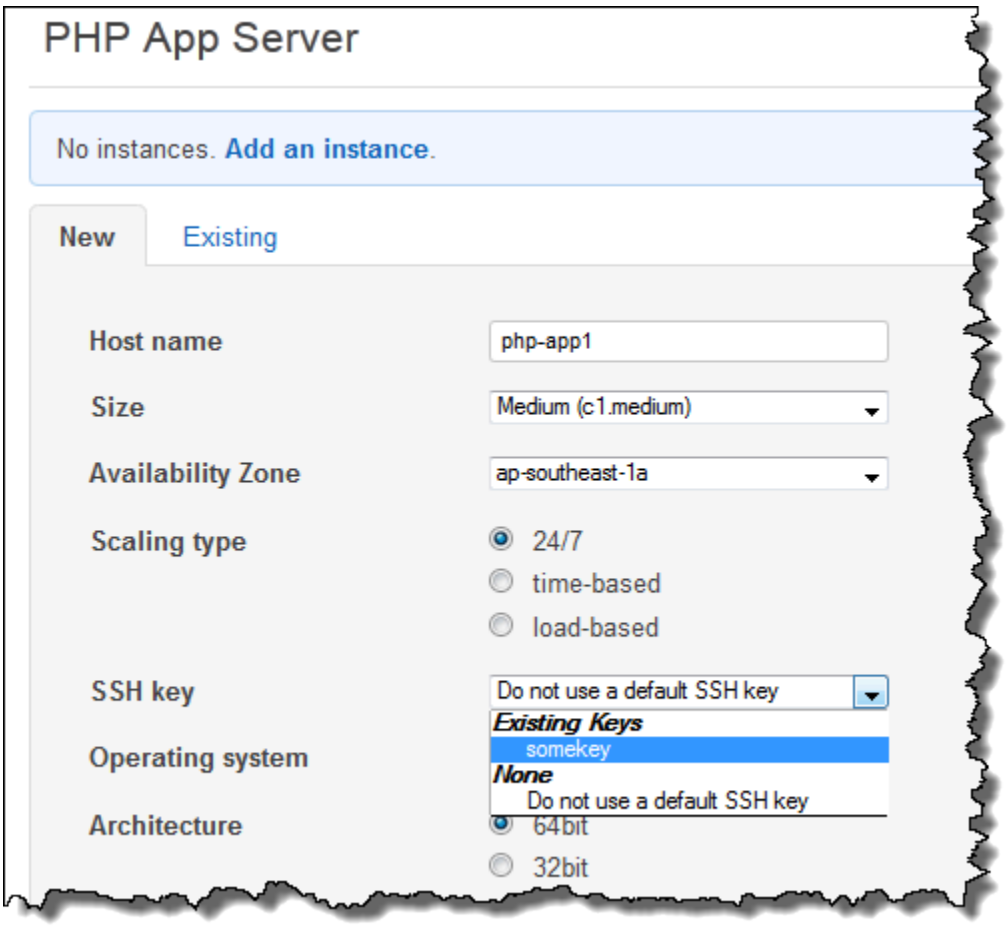
When you create a stack, you can specify an Amazon EC2 SSH key that is installed by default on every instance in the stack.



The **Default SSH key** list shows your AWS account's Amazon EC2keys. You can do one of the following:

- Select the appropriate key from the list.
- Select **Do not use a default SSH key** to specify no key.

If you selected **Do not use a default SSH key**, or you want to override a stack's default key, you can specify a key when you create an instance.



When you start the instance OpsWorks Stacks installs the public key in the `authorized_keys` file.

Registering a User's Public SSH Key

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

There are two ways to register a user's public SSH key:

- An administrative user can assign a public SSH key to one or more users and provide them with the corresponding private key.

- An administrative user can enable self-management for one or more users.

  Those users can then specify their own public SSH key.

For more information how administrative users can enable self management or assign public keys to users, see Editing User Settings.

Connecting to Linux-based instances by using SSH in a PuTTY terminal requires additional steps. For more information, see Connecting to Your Linux Instance from Windows Using PuTTY and Troubleshooting Connecting to Your Instance in the AWS documentation.

The following describes how a user with self-management enabled can specify their public key.

To specify your SSH public key

1. Create an SSH key pair.

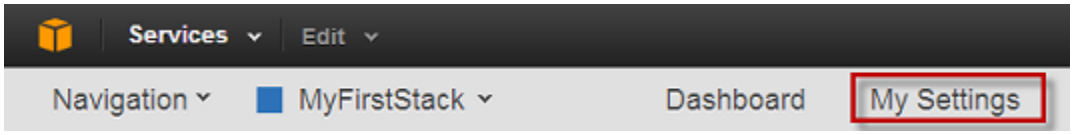
   The simplest approach is to generate the key pair locally. For more information see How to Generate Your Own Key and Import It to Amazon EC2.

   > **Note**
   >
   > If you use PuTTYgen to generate your key pair, copy the public key from the **Public key for pasting into OpenSSH authorized_keys file** box. Clicking **Save Public Key** saves the public key in a format that is not supported by MindTerm.

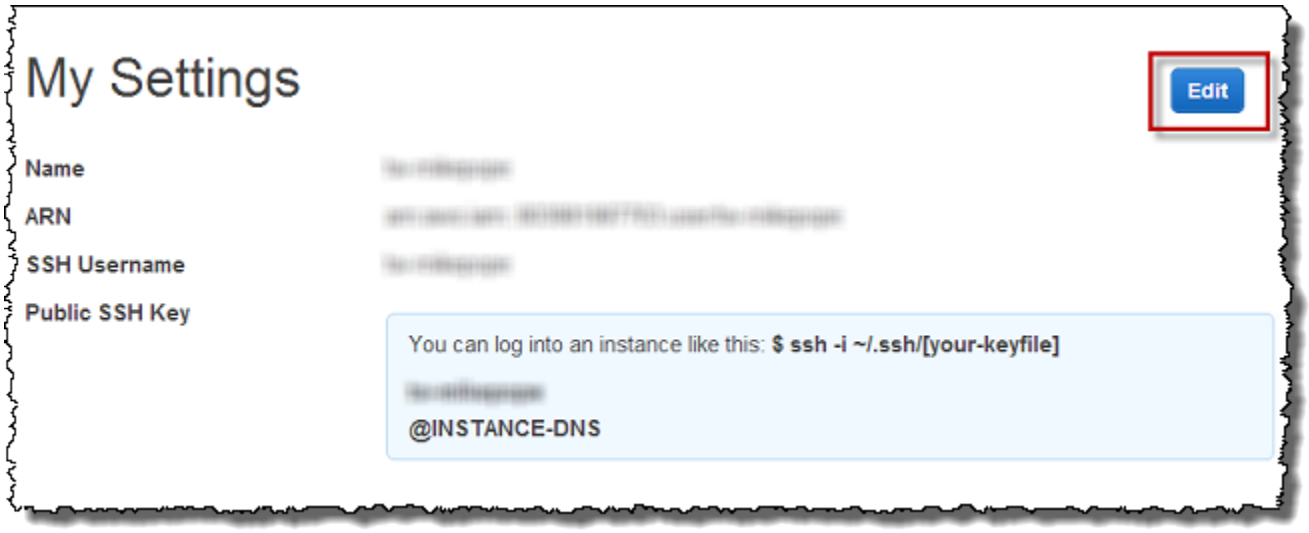
2. Sign into the OpsWorks Stacks console as an IAM user with self-management enabled.

   > **Important**
   >
   > If you sign in as an account owner, or as an IAM user that does not have self-management enabled, OpsWorks Stacks does not display **My Settings**. If you are an administrative user or the account owner, you can instead specify SSH keys by going to the **Users** page and editing the user settings.

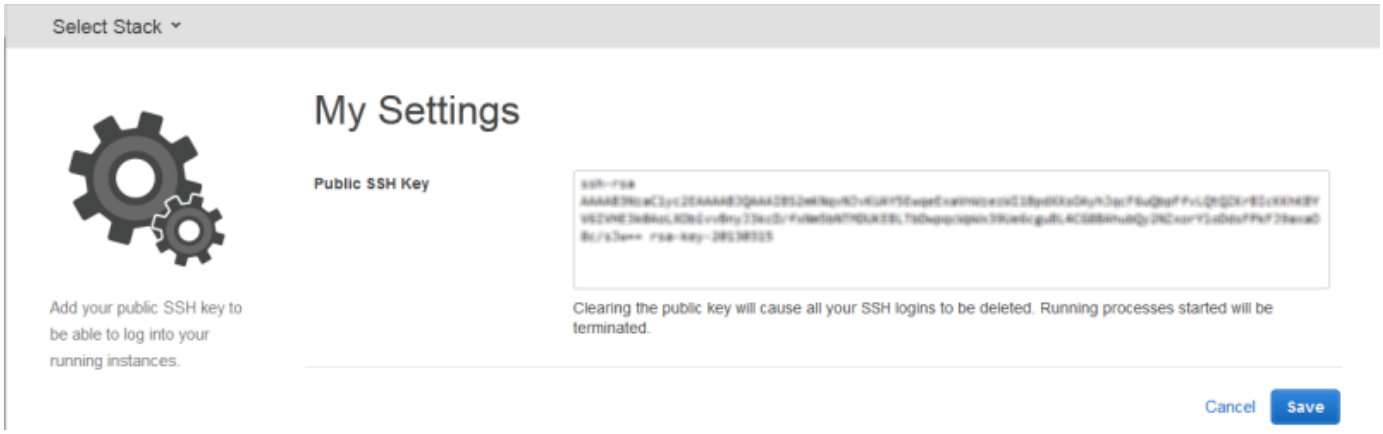
3. Select **My Settings**, which displays the settings for the signed-in user.



4. On the **My Settings** page, click **Edit**.



5. In the **Public SSH Key box**, enter your SSH public key, and then click **Save**.



Important

To use the built-in MindTerm SSH client to connect to Amazon EC2 instances, a user must be signed in as an IAM user and have a public SSH key registered with OpsWorks Stacks. For more information, see Using the Built-in MindTerm SSH Client.

Managing Linux Security Updates

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Security Updates

Linux operating system providers supply regular updates, most of which are operating system security patches but can also include updates to installed packages. You should ensure that your instances' operating systems are current with the latest security patches.

By default, OpsWorks Stacks automatically installs the latest updates during setup, after an instance finishes booting. OpsWorks Stacks does not automatically install updates after an instance is online, to avoid interruptions such as restarting application servers. Instead, you manage updates to your online instances yourself, so you can minimize any disruptions.

We recommend that you use one of the following to update your online instances.

- Create and start new instances to replace your current online instances. Then delete the current instances.

  The new instances will have the latest set of security patches installed during setup.
- On Linux-based instances in Chef 11.10 or older stacks, run the Update Dependencies stack command, which installs the current set of security patches and other updates on the specified instances.

For both of these approaches, OpsWorks Stacks performs the update by running `yum update` for Amazon Linux and Red Hat Enterprise Linux (RHEL) or `apt-get update` for Ubuntu. Each distribution handles updates somewhat differently, so you should examine the information in the associated links to understand exactly how an update will affect your instances:

- **Amazon Linux** – Amazon Linux updates install security patches and might also install feature updates, including package updates.

  For more information, see Amazon Linux AMI FAQs.
- **Ubuntu** – Ubuntu updates are largely limited to installing security patches, but might also install package updates for a limited number of critical fixes.

  For more information, see LTS - Ubuntu Wiki.
- **CentOS** – CentOS updates generally maintain binary compatibility with earlier versions.
- **RHEL** – RHEL updates generally maintain binary compatibility with earlier versions.

  For more information, see Red Hat Enterprise Linux Life Cycle.

If you want more control over updates, such as specifying particular package versions, you can disable automatic updates by using the CreateInstance, UpdateInstance, CreateLayer, or UpdateLayer actions—or the equivalent AWS SDK methods or AWS CLI commands—to set the `InstallUpdatesOnBoot` parameter to `false`. The following example shows how to use the AWS CLI to disable `InstallUpdatesOnBoot` as the default setting for an existing layer.

```
aws opsworks update-layer --layer-id layer ID --no-install-updates-on-boot
```

You must then manage updates yourself. For example, you could employ one of these strategies:

- Implement a custom recipe that runs the appropriate shell command to install your preferred updates.

  Because system updates don't map naturally to a lifecycle event, include the recipe in your custom cookbooks but execute it manually. For package updates, you can also use the yum_package (Amazon Linux) or apt_package (Ubuntu) resources instead of a shell command.
- Log in to each instance with SSH and run the appropriate commands manually.

Using Security Groups

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate

their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Security Groups

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Each Amazon EC2 instance has one or more associated security groups that govern the instance's network traffic, much like a firewall. A security group has one or more *rules*, each of which specifies a particular category of allowed traffic. A rule specifies the following:

- The type of allowed traffic, such as SSH or HTTP
- The traffic's protocol, such as TCP or UDP
- The IP address range that the traffic can originate from
- The traffic's allowed port range

Security groups have two types of rules:

- Inbound rules govern inbound network traffic.

  For example, application server instances commonly have an inbound rule that allows inbound HTTP traffic from any IP address to port 80, and another inbound rule that allows inbound SSH traffic to port 22 from specified set of IP addresses.
- Outbound rules govern outbound network traffic.

  A common practice is to use the default setting, which allows any outbound traffic.

For more information about security groups, see Amazon EC2 Security Groups.

The first time you create a stack in a region, OpsWorks Stacks creates a built-in security group for each layer with an appropriate set of rules. All of the groups have default outbound rules, which allow all outbound traffic. In general, the inbound rules allow the following:

- Inbound TCP, UDP, and ICMP traffic from the appropriate OpsWorks Stacks layers
- Inbound TCP traffic on port 22 (SSH login)

> **Warning**
>
> The default security group configuration opens SSH (port 22) to any network location (0.0.0.0/0.) This allows all IP addresses to access your instance by using SSH. For production environments, you must use a configuration that only allows SSH access from a specific IP address or range of addresses. Either update the default security groups immediately after they are created, or use custom security groups instead.

- For web server layers, all inbound TCP, and UDP traffic to ports 80 (HTTP) and 443 (HTTPS)

> **Note**
>
> The built-in `AWS-OpsWorks-RDP-Server` security group is assigned to all Windows instances to allow RDP access. However, by default, it does not have any rules. If you are running a Windows stack and want to use RDP to access instances, you must add an inbound rule that allows RDP access. For more information, see Logging In with RDP.

To see the details for each group, go to the Amazon EC2 console, select **Security Groups** in the navigation pane, and select the appropriate layer's security group. For example, **AWS-OpsWorks-Default-Server** is the default built-in security group for all stacks, and **AWS-OpsWorks-WebApp** is the default built-in security group for the Chef 12 sample stack.

> **Note**
>
> If you accidentally delete an OpsWorks Stacks security group, the preferred way to recreate it is to have OpsWorks Stacks perform the task for you. Just create a new stack in the same AWS region—and VPC, if present—and OpsWorks Stacks will automatically recreate all the built-in security groups, including the one that you deleted. You can then delete the stack if you don't have any further use for it; the security groups will remain. If you

want to recreate the security group manually, it must be an exact duplicate of the original, including the group name's capitalization.
Additionally, OpsWorks Stacks will attempt to recreate all built-in security groups if any of the following occur:

- You make any changes to the stack's settings page in the OpsWorks Stacks console.
- You start one of the stack's instances.
- You create a new stack.

You can use either of the following approaches for specifying security groups. You use the **Use OpsWorks security groups** setting to specify your preference when you create a stack.

- **Yes** (default setting) – OpsWorks Stacks automatically associates the appropriate built-in security group with each layer.

  You can fine-tune a layer's built-in security group by adding a custom security group with your preferred settings. However, when Amazon EC2 evaluates multiple security groups, it uses the least restrictive rules, so you cannot use this approach to specify more restrictive rules than the built-in group.
- **No** – OpsWorks Stacks does not associate built-in security groups with layers.

  You must create appropriate security groups and associate at least one with each layer that you create. Use this approach to specify more restrictive rules than the built-in groups. Note that you can still manually associate a built-in security group with a layer if you prefer; custom security groups are required only for those layers that need custom settings.

Important

If you use the built-in security groups, you cannot create more restrictive rules by manually modifying the group's settings. Each time you create a stack, OpsWorks Stacks overwrites the built-in security groups' configurations, so any changes that you make will be lost the next time you create a stack. If a layer requires more restrictive security group settings than the built-in security group, set **Use OpsWorks security groups** to **No**, create custom security groups with your preferred settings, and assign them to the layers on creation.

OpsWorks Stacks Support for Chef 12 Linux

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section provides a brief overview of OpsWorks Stacks for Chef 12 Linux. For information about Chef 12 on Windows, see Getting Started: Windows. For information about previous Chef versions on Linux, see Chef 11.10 and Earlier Versions for Linux.

Overview

OpsWorks Stacks supports Chef 12, the latest version of Chef, for Linux stacks. For more information, see Learn Chef.

OpsWorks Stacks continues to support Chef 11.10 for Linux stacks. However, if you are an advanced Chef user who would like to benefit from the large selection of community cookbooks or write your own custom cookbooks, we recommend that you use Chef 12. Chef 12 stacks provide the following advantages over Chef 11.10 and earlier stacks for Linux:

- **Two separate Chef runs** - When a command is executed on an instance, the OpsWorks Stacks agent now executes two isolated Chef runs: one run for tasks that integrate the instance with other AWS services like AWS Identity and Access Management (IAM), and one run for your custom cookbooks. The first Chef run installs the OpsWorks Stacks agent on the instance and performs system tasks such as user setup and management, volume setup and configuration, configuration of CloudWatch metrics, and so on. The second run is dedicated exclusively to running your custom recipes for OpsWorks Stacks Lifecycle Events. This second run lets you use your own Chef cookbooks or community cookbooks.
- **Resolution of namespace conflicts** - Prior to Chef 12, OpsWorks Stacks performed system tasks and ran built-in and custom recipes in a shared environment. This resulted in namespace conflicts and lack of clarity about which recipes OpsWorks Stacks had run. Unwanted default configurations had to be manually overwritten, a time consuming and error-prone task. In Chef 12 for Linux, OpsWorks Stacks no longer supports built-in Chef cookbooks for application

server environments like PHP, Node.js, or Rails. By eliminating built-in recipes, OpsWorks Stacks eliminates the issue of naming collisions between built-in and custom recipes.

- **Strong support for Chef community cookbooks** – OpsWorks Stacks Chef 12 Linux offers greater compatibility and support for community cookbooks from the Chef supermarket. You can now use community cookbooks that are superior to the built-in cookbooks that OpsWorks Stacks previously provided—cookbooks that are designed for use with the latest application server environments and frameworks. You can run most of these cookbooks without modification on Chef 12 for Linux. For more information, go to Chef Supermarket on the Learn Chef website, the Chef Supermarket website, and the Chef Cookbooks repository on GitHub.
- **Timely Chef 12 updates** - OpsWorks Stacks will update its Chef environment to the latest Chef 12 version shortly after each Chef release. With Chef 12, minor Chef updates and new OpsWorks Stacks agent releases will coincide. This lets you test new Chef releases directly, and enables your Chef recipes and applications to take advantage of the latest Chef features.

For more information about supported Chef versions prior to Chef 12, see Chef 11.10 and Earlier Versions for Linux.

Moving to Chef 12

Key OpsWorks Stacks changes for Chef 12 Linux, as compared to support for previous Chef versions 11.10, 11.4, and 0.9, are as follows:

- Built-in layers are no longer provided or supported for Chef 12 for Linux stacks. Because only your custom recipes are executed, removing this support gives total transparency into how the instance is set up and makes custom cookbooks much easier to write and maintain. For example, it's no longer necessary to overwrite attributes of built-in OpsWorks Stacks recipes. Removal of built-in layers also enables OpsWorks Stacks to better support cookbooks that are developed and maintained by the Chef community, so that you can take full advantage of them. The built-in layer types no longer available in Chef 12 for Linux are: AWS Flow (Ruby), Ganglia, HAProxy, Java App Server, Memcached, MySQL, Node.js App Server, PHP App Server, Rails App Server, and Static Web Server.
  - Because OpsWorks Stacks is running recipes that you provide, there is no longer a need to override built-in OpsWorks Stacks attributes by running custom cookbooks. To override attributes in your own or community recipes, follow instructions and examples in About Attributes in the Chef 12 documentation.
- OpsWorks Stacks continues to provide support for the following layers for Chef 12 Linux stacks:

  - Custom Layers
  - Amazon RDS Service Layer
  - ECS Cluster Layers
- Stack configuration and data bags for Chef 12 Linux have changed to look very similar to their counterparts for Chef 12.2 Windows. This makes it easier to query for, analyze, and troubleshoot these data bags, especially if you work with stacks with different operating system types. Note that OpsWorks Stacks does not support encrypted data bags. To store sensitive data in encrypted form, such as passwords or certificates, we recommend storing it in a private S3 bucket. You can then create a custom recipe that uses the Amazon SDK for Ruby to retrieve the data. For an example, see Using the SDK for Ruby.For more information, see OpsWorks Stacks Data Bag Reference.
- In Chef 12 Linux, Berkshelf is no longer installed on stack instances. Instead, we recommend that you use Berkshelf on a local development machine to package your cookbook dependencies locally. Then upload your package, with the dependencies included, to Amazon Simple Storage Service. Finally, modify your Chef 12 Linux stack to use the uploaded package as a cookbook source. For more information, see Packaging Cookbook Dependencies Locally.
- RAID configurations for EBS volumes are no longer supported. For increased performance, you can use provisioned IOPS for Amazon Elastic Block Store (Amazon EBS).
- autofs is no longer supported.
- Subversion repositories are no longer supported.
- Per-layer OS package installations must now be done with custom recipes. For more information, see Per-layer Package Installations.

Supported Operating Systems

Chef 12 supports the same Linux operating systems as previous versions of Chef. For a list of Linux operating system types and versions that Chef 12 Linux stacks can use, see Linux operating systems.

Supported Instance Types

OpsWorks Stacks supports all instance types for Chef 12 Linux stacks except specialized instance types like high performance computing (HPC) cluster compute, cluster GPU, and high memory cluster instance types.

More Information

To learn more about how to work with Chef 12 for Linux stacks, see the following:

- Getting Started: Sample

  Introduces you to OpsWorks Stacks by guiding you through a brief hands-on exercise with the OpsWorks Stacks console to create a Node.js application environment.

- Getting Started: Linux

  Introduces you to OpsWorks Stacks and Chef 12 Linux by guiding you through a hands-on exercise with the OpsWorks Stacks console to create a basic Chef 12 Linux stack that contains a simple layer with a Node.js app that serves traffic.

- Custom Layers

  Provides guidance for adding a layer that contains cookbooks and recipes to a Chef 12 Linux stack. You can use readily available cookbooks and recipes that the Chef community provides, or you can create your own.

- Moving to Data Bags

  Compares and contrasts instance JSON that is used by Linux stacks running Chef 11 and earlier versions with Chef 12. Also provides pointers to reference documentation for the Chef 12 instance JSON format.

Moving Stack Settings from Attributes to Data Bags

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks exposes a wide variety of stack settings to your Chef recipes. These stack settings include values such as:

- Stack cookbook source URLs
- Layer volume configurations
- Instance host names
- Elastic Load Balancing DNS names
- App source URLs
- User names

Referencing stack settings from recipes makes recipe code more robust and less error prone than hard-coding stack settings directly in recipes. This topic describes how to access these stack settings as well as how to move from attributes in Chef 11.10 and earlier versions for Linux to data bags in Chef 12 Linux.

In Chef 11.10 and earlier versions for Linux, stack settings are available as Chef attributes and are accessed through the Chef `node` object or through Chef search. These attributes are stored on OpsWorks Stacks instances in a set of JSON files in the `/var/lib/aws/opsworks/chef` directory. For more information, see Stack Configuration and Deployment Attributes: Linux.

In Chef 12 Linux, stack settings are available as Chef data bags and are accessed only through Chef search. Data bags are stored on OpsWorks Stacks instances in a set of JSON files in the `/var/chef/runs/`*`run-ID`*`/data_bags` directory, where *`run-ID`* is a unique ID that OpsWorks Stacks assigns to each Chef run on an instance. Stack settings are no longer available as Chef attributes, so stack settings can no longer be accessed through the Chef `node` object. For more information, see the OpsWorks Stacks Data Bag Reference.

For example, in Chef 11.10 and earlier versions for Linux, the following recipe code uses the Chef `node` object to get attributes representing an app's short name and source URL. It then uses the Chef log to write these two attribute values:

```
Chef::Log.info ("********** The app's short name is '#{node['opsworks']
['applications'].first['slug_name']}' **********")
Chef::Log.info("********** The app's URL is '#{node['deploy']['simplephpapp']['scm']
['repository']}' **********")
```

In Chef 12 Linux, the following recipe code uses the `aws_opsworks_app` search index to get the contents of the first data bag item in the `aws_opsworks_app` data bag. The code then writes two messages to the Chef log, one with the app's short name data bag content, and another with the app's source URL data bag content:

```
app = search("aws_opsworks_app").first

Chef::Log.info("********** The app's short name is '#{app['shortname']}' **********")
Chef::Log.info("********** The app's URL is '#{app['app_source']['url']}' **********")
```

To migrate your recipe code that accesses stack settings from Chef 11.10 and earlier versions for Linux to Chef 12 Linux, you must revise your code to:

- Access Chef data bags instead of Chef attributes.
- Use Chef search instead of the Chef `node` object.
- Use OpsWorks Stacks data bag names such as `aws_opsworks_app`, instead of using OpsWorks Stacks attribute names such as `opsworks` and `deploy`.

For more information, see the OpsWorks Stacks Data Bag Reference.

Support for Previous Chef Versions in OpsWorks Stacks

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section provides a brief overview of the OpsWorks Stacks documentation for previous Chef versions.

Chef 11.10 and Earlier Versions for Linux

Provides documentation about OpsWorks Stacks support for Chef 11.10, 11.4, and 0.9 for Linux stacks.

Chef 11.10 and Earlier Versions for Linux

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section provides a brief overview of the OpsWorks Stacks documentation for Chef 11.10, 11.4, and 0.9 for Linux.

Getting Started with Chef 11 Linux Stacks

Provides a walkthrough that shows you how to create a simple but functional PHP application server stack.

Creating Your First Node.js Stack

Describes how to create a Linux stack that supports a Node.js application server and how to deploy a simple application.

Customizing OpsWorks Stacks

Describes how to customize OpsWorks Stacks to meet your specific requirements.

Cookbooks 101

Describes how to implement recipes for OpsWorks Stacks instances.

Load Balancing a Layer

Describes how to use available OpsWorks Stacks load balancing options.

Running a Stack in a VPC

Describes how to create and run a stack in a virtual private cloud.

Migrating from Chef Server

Provides guidelines for migrating from Chef Server to OpsWorks Stacks.

[OpsWorks Stacks Layer Reference]

Describes the available OpsWorks Stacks built-in layers.

[Cookbook Components]

Describes the three standard cookbook components: attributes, templates, and recipes.

[Stack Configuration and Deployment Attributes: Linux]

Describes stack configuration and deployment attributes for Linux.

[Built-in Cookbook Attributes]

Describes how to use built-in recipe attributes to control the configuration of installed software.

[Troubleshooting Chef 11.10 and Earlier Versions for Linux]

Describes approaches to troubleshooting various issues in OpsWorks Stacks.

Getting Started with Chef 11 Linux Stacks

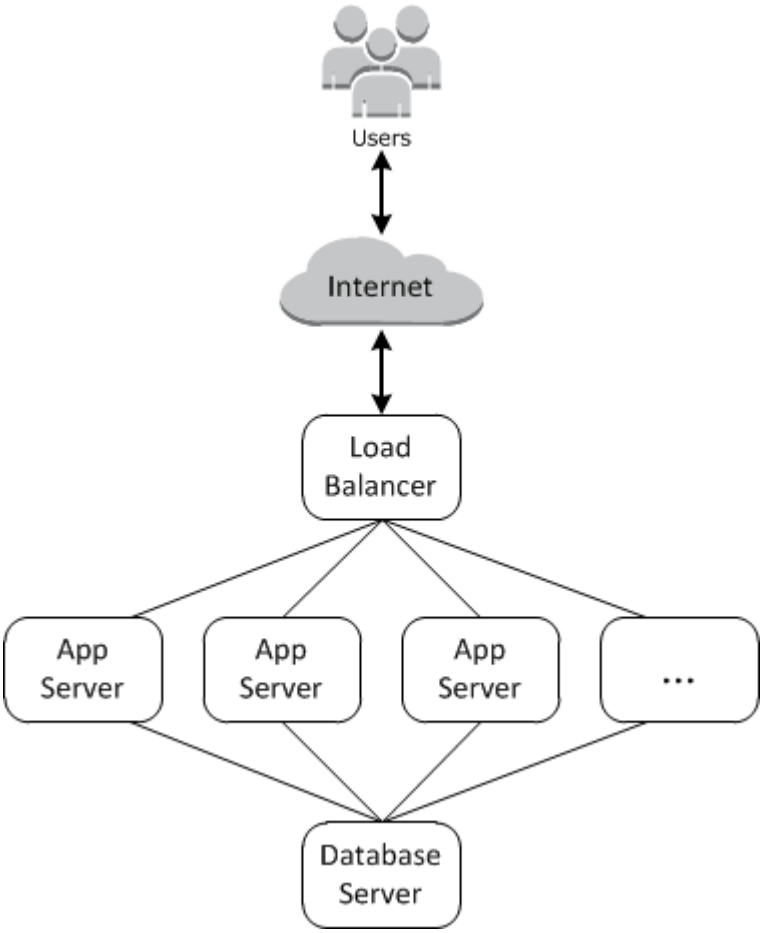
Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

This section describes how to get started with Linux stacks using Chef 11. For information about getting started with Chef 12 Linux stacks, see Getting Started: Linux. For information about getting started with Chef 12 Windows stacks, see Getting Started: Windows.

Cloud-based applications usually require a group of related resources—application servers, database servers, and so on—that must be created and managed collectively. This collection of instances is called a *stack*. A simple application stack might look something like the following.



The basic architecture consists of the following:

- A load balancer to distribute incoming traffic from users evenly across the application servers.
- A set of application server instances, as many as needed to handle the traffic.
- A database server to provide the application servers with a back-end data store.

In addition, you typically need a way to distribute applications to the application servers, monitor the stack, and so on.

OpsWorks Stacks provides a simple and straightforward way to create and manage stacks and their associated applications and resources. This chapter introduces the basics of OpsWorks Stacks—along with some of its more sophisticated features—by walking you through the process of creating the application server stack in the diagram. It uses an incremental development model that OpsWorks Stacks makes easy to follow: Set up a basic stack and, once it's working correctly, add components until you arrive at a full-featured implementation.

- Step 1: Complete the Prerequisites shows how to get set up to start the walkthrough.

- Step 2: Create a Simple Application Server Stack - Chef 11 shows how to create a minimal stack that consists of a single application server.
- Step 3: Add a Back-end Data Store shows how to add a database server and connect it to the application server.
- Step 4: Scale Out MyStack shows how to scale out a stack to handle increased load by adding more application servers, and a load balancer to distribute incoming traffic.

Topics

- Step 1: Complete the Prerequisites
- Step 2: Create a Simple Application Server Stack - Chef 11
- Step 3: Add a Back-end Data Store
- Step 4: Scale Out MyStack
- Step 5: Delete MyStack

Step 1: Complete the Prerequisites

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Complete the following setup steps before you can start the walkthrough. These setup steps include signing up for an AWS account, creating an administrative user, and assigning access permissions to OpsWorks Stacks.

If you have already completed any of the Getting Started with OpsWorks Stacks walkthroughs, then you have met the prerequisites for this walkthrough, and you can skip ahead to Step 2: Create a Simple Application Server Stack - Chef 11.

Topics

- Sign up for an AWS account

- Create a user with administrative access
- Assign Service Access Permissions to Your User

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.
2. Turn on multi-factor authentication (MFA) for your root user.

For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Assign Service Access Permissions to Your User

Enable access to the OpsWorks Stacks service (and related services that OpsWorks Stacks relies on) by adding the `AWSOpsWorks_FullAccess` and `AmazonS3FullAccess` permissions to your role or user.

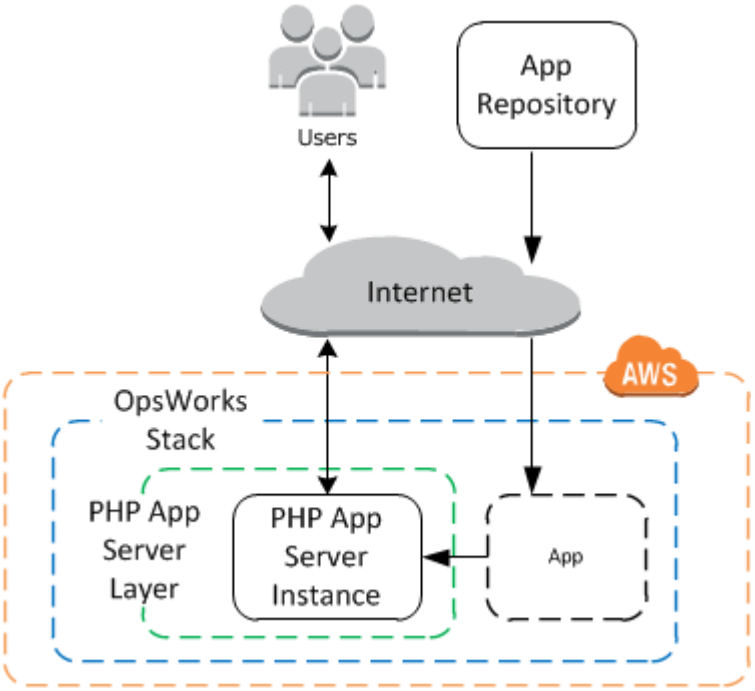
For more information about adding permissions, see Adding IAM identity permissions (console).

You have now completed all of the setup steps and can start this walkthrough.

Step 2: Create a Simple Application Server Stack - Chef 11

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

A basic application server stack consists of a single application server instance with a public IP address to receive user requests. Application code and any related files are stored in a separate repository and deployed from there to the server. The following diagram illustrates such a stack.



The stack has the following components:

- A *layer*, which represents a group of instances and specifies how they are to be configured.

  The layer in this example represents a group of PHP App Server instances.
- An *instance*, which represents an Amazon EC2 instance.

  In this case, the instance is configured to run a PHP app server. Layers can have any number of instances. OpsWorks Stacks also supports several other app servers. For more information, see Application Server Layers.
- An *app*, which contains the information required to install an application on the application server.

  The code is stored in a remote repository, such as Git repository or an Amazon S3 bucket.

The following sections describe how to use the OpsWorks Stacks console to create the stack and deploy the application. You can also use an AWS CloudFormation template to provision a stack. For an example template that provisions the stack described in this topic, see AWS OpsWorks Snippets.

Topics

- Step 2.1: Create a Stack - Chef 11
- Step 2.2: Add a PHP App Server Layer - Chef 11
- Step 2.3: Add an Instance to the PHP App Server Layer - Chef 11
- Step 2.4: Create and Deploy an App - Chef 11

Step 2.1: Create a Stack - Chef 11

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You start an OpsWorks Stacks project by creating a stack, which acts as a container for your instances and other resources. The stack configuration specifies some basic settings, such as the AWS region and the default operating system, that are shared by all the stack's instances.

> **Note**
>
> This page helps you create Chef 11 stacks. For information about how to create Chef 12 stacks, see Create a Stack.

This page helps you create stacks in Chef 11.

To create a new stack

1. **Add a Stack**

   Sign into the OpsWorks Stacks console. If the account has no existing stacks, you will see the **Welcome to AWS OpsWorks** page; click **Add your first stack**. Otherwise, you will see the OpsWorks Stacks dashboard, which lists your account's stacks; click **Add Stack**.



2. **Configure the Stack**

   On the **Add Stack** page, choose **Chef 11 stack** and specify the following settings:

   Stack name

   Enter a name for your stack, which can contain alphanumeric characters (a–z, A–Z, and 0–9), and hyphens (-). The example stack for this walkthrough is named **`MyStack`**.

Region

Select US West (Oregon) as the stack's region.

Accept the default values for the other settings and click **Add Stack**. For more information on the various stack settings, see Create a New Stack.

Step 2.2: Add a PHP App Server Layer - Chef 11

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Although a stack is basically a container for instances, you don't add instances directly to a stack. You add a layer, which represents a group of related instances, and then add instances to the layer.

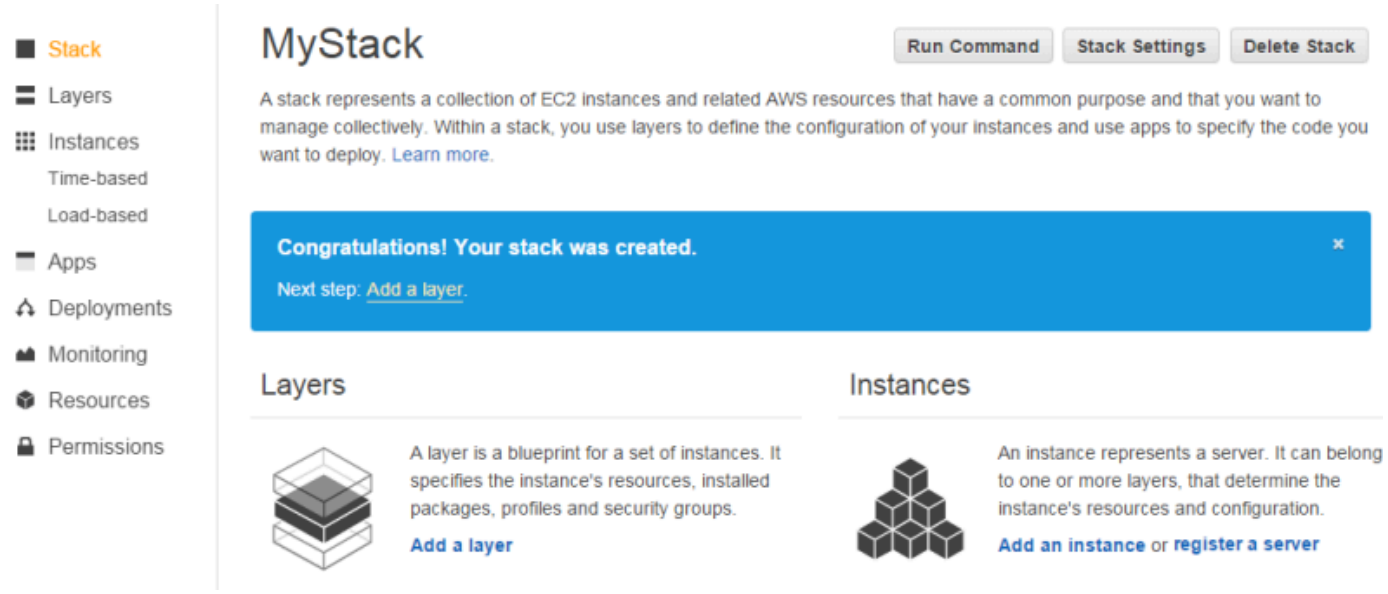
A layer is basically a blueprint that OpsWorks Stacks uses to create set of Amazon EC2 instances with the same configuration. You add one layer to the stack for each group of related instances. OpsWorks Stacks includes a set of built-in layers to represent groups of instances running standard software packages such as a MySQL database server or a PHP application server. In addition, you can create partially or fully customized layers to suit your specific requirements. For more information, see Customizing OpsWorks Stacks.

MyStack has one layer, the built-in PHP App Server layer, which represents a group of instances that function as PHP application servers. For more information, including descriptions of the built-in layers, see Layers.

To add a PHP App Server layer to MyStack

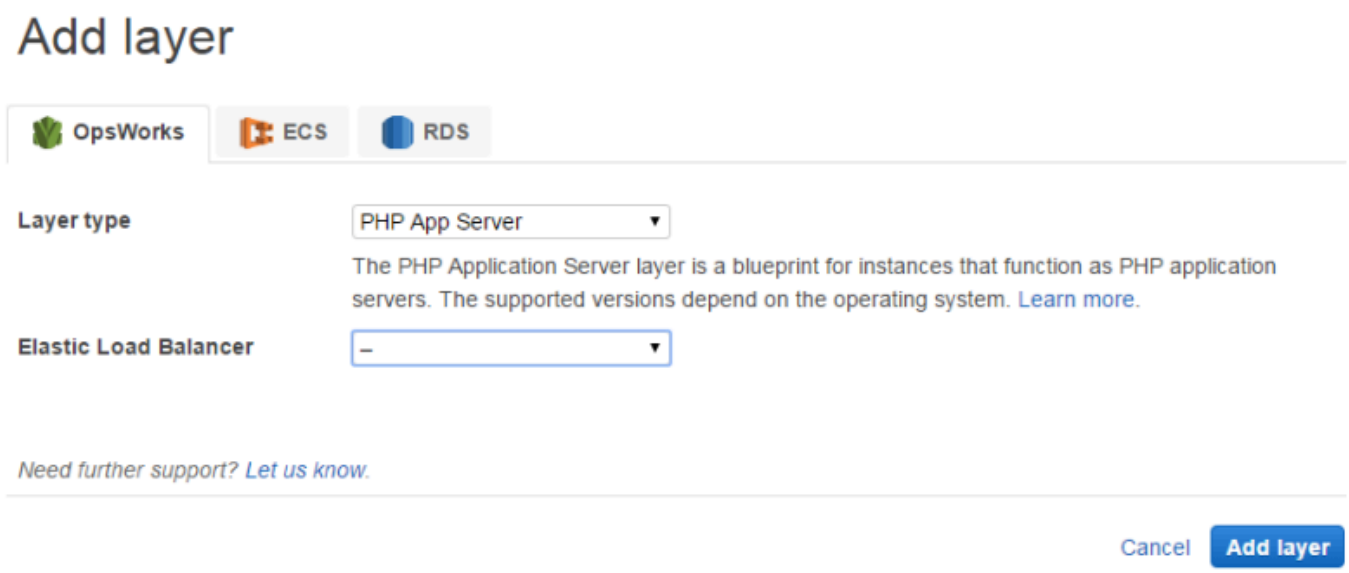
1. **Open the Add Layer Page**

   After you finish creating the stack, OpsWorks Stacks displays the **Stack** page. Click **Add a layer** to add your first layer.



2. **Specify a Layer Type and Configure the Layer**

   In the **Layer type** box, select **PHP App Server**, accept the default **Elastic Load Balancer** setting and click **Add Layer**. After you create the layer, you can specify other attributes such as the EBS volume configuration by editing the layer.



Step 2.3: Add an Instance to the PHP App Server Layer - Chef 11

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

An OpsWorks Stacks instance represents a particular Amazon EC2 instance:

- The instance's configuration specifies some basics like the Amazon EC2operating system and size; it runs but doesn't do very much.
- The instance's layer adds functionality to the instance by determining which packages are to be installed, whether the instance has an Elastic IP address, and so on.

OpsWorks Stacks installs an agent on each instance that interacts with the service. To add a layer's functionality to an instance, OpsWorks Stacks directs the agent to run small applications called Chef recipes, which can install applications and packages, create configuration files, and so on. OpsWorks Stacks runs recipes at key points in the instance's lifecycle. For example, OpsWorks runs Setup recipes after the instance has finished booting to handle tasks such as installing software, and runs Deploy recipes when you deploy an app to install the code and related files.

> **Note**
>
> If you are curious about how the recipes work, all of the OpsWorks Stacks built-in recipes are in a public GitHub repository: OpsWorks Cookbooks. You can also create your own custom recipes and have OpsWorks Stacks run them, as described later.

To add a PHP application server to MyStack, add an instance to the PHP App Server layer that you created in the previous step.

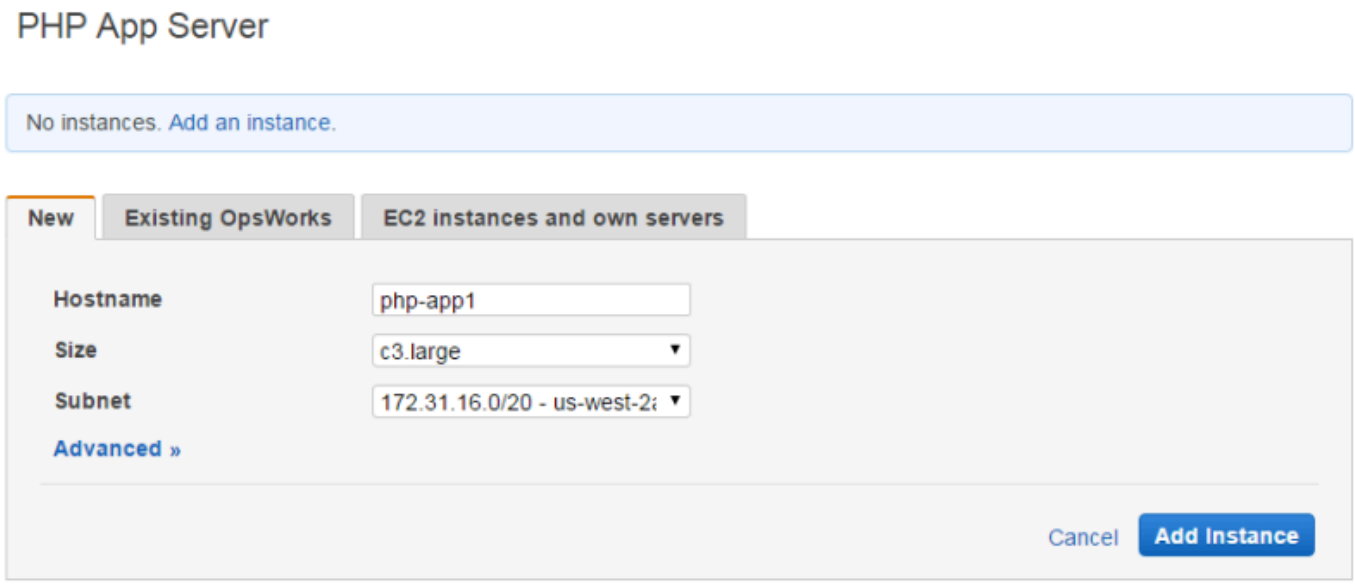
To add an instance to the PHP App Server layer

1. **Open Add an Instance**

   After you finish adding the layer, OpsWorks Stacks displays the **Layers** page. Click **Instances** in the navigation pane and under **PHP App Server**, click **Add an instance**.

2. **Configure the Instance**

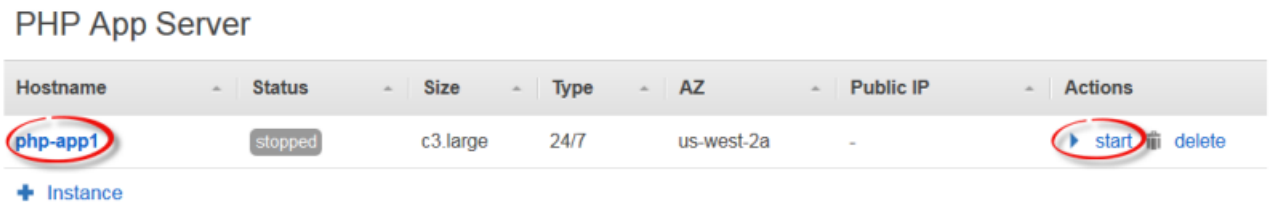
   Each instance has a default host name that is generated for you by OpsWorks Stacks. In this example, OpsWorks Stacks simply adds a number to the layer's short name. You can configure each instance separately, including overriding some of the default settings that you specified when creating the stack, such as the Availability Zone or operating system. For this walkthrough, just accept the default settings and click **Add Instance** to add the instance to the layer. For more information, see Instances.



3. **Start the Instance**

   So far, you have just specified the instance's configuration. You have to start an instance to create a running Amazon EC2 instance. OpsWorks Stacks then uses the configuration settings to launch an Amazon EC2 instance in the specified Availability Zone. The details of how you start an instance depend on the instance's *scaling type*. In the previous step, you created an instance with the default scaling type, *24/7*, which must be manually started and then runs until it is manually stopped. You can also create time-based and load-based scaling types, which OpsWorks Stacks automatically starts and stops based on a schedule or the current load. For more information, see Managing load with time-based and load-based instances.

Go to **php-app1** under **PHP App Server** and click **start** in the row's **Actions** column to start the instance.

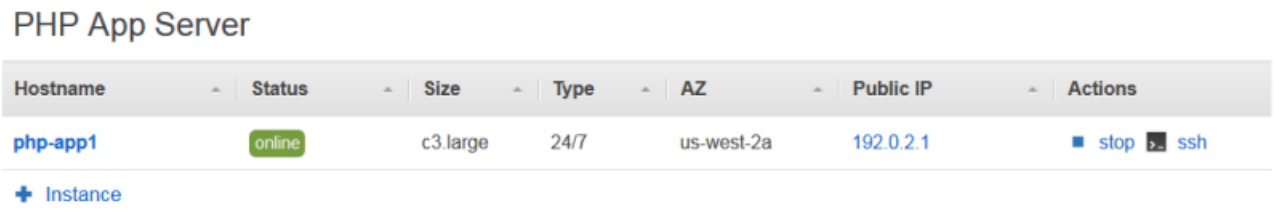


4. **Monitor the Instance's Status during Startup**

   It typically takes a few minutes to boot the Amazon EC2 instance and install the packages. As startup progresses, the instance's **Status** field displays the following series of values:

   1. **requested** - OpsWorks Stacks has called the Amazon EC2 service to create the Amazon EC2 instance.
   2. **pending** - OpsWorks Stacks is waiting for the Amazon EC2 instance to start.
   3. **booting** - The Amazon EC2 instance is booting.
   4. **running_setup** - The OpsWorks Stacks agent is running the layer's Setup recipes, which handle tasks such as configuring and installing packages, and the Deploy recipes, which deploy any apps to the instance.
   5. **online** - The instance is ready for use.

   After php-app1 comes online, the **Instances** page should look like the following:



   The page begins with a quick summary of all your stack's instances. Right now, it shows one online instance. In the php-app1 **Actions** column, notice that **stop**, which stops the instance, has replaced **start** and **delete**.

Step 2.4: Create and Deploy an App - Chef 11

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

To make MyStack more useful, you need to deploy an app to the PHP App Server instance. You store an app's code and any related files in a repository, such as Git. You need to take a couple of steps to get those files to your application servers:

> **Note**
>
> The procedure in this section applies to Chef 11 stacks. For information about how to add apps to layers in Chef 12 stacks, see Adding Apps.

1. Create an app.

   An app contains the information that OpsWorks Stacks needs in order to download the code and related files from the repository. You can also specify additional information such as the app's domain.
2. Deploy the app to your application servers.

   When you deploy an app, OpsWorks Stacks triggers a Deploy lifecycle event. The agent then runs the instance's Deploy recipes, which download the files to the appropriate directory along with related tasks such as configuring the server, restarting the service, and so on.

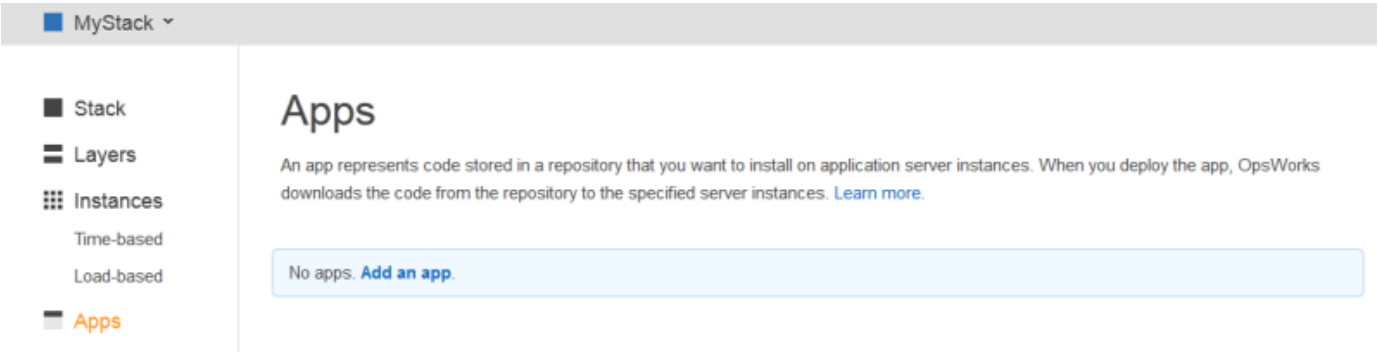
> **Note**
>
> When you create a new instance, OpsWorks Stacks automatically deploys any existing apps to the instance. However, when you create a new app or update an existing one, you must manually deploy the app or update to all existing instances.

This step shows how to manually deploy an example app from a public Git repository to an application server. If you would like to examine the application, go to https://github.com/amazonwebservices/opsworks-demo-php-simple-app. The application used in this example is in the version1 branch. OpsWorks Stacks also supports several other repository types. For more information, see Application Source.

To create and deploy an app

1. **Open the Apps Page**

   In the navigation pane, click **Apps** and on the **Apps** page, click **Add an app**.



2. **Configure the App**

   On the **App** page, specify the following values:

   Name

   The app's name, which OpsWorks Stacks uses for display purposes. The example app is named **`SimplePHPApp`**. OpsWorks Stacks also generates a short name—simplephpapp for this example—that is used internally and by the Deploy recipes, as described later.

   Type

   The app's type, which determines where to deploy the app. The example uses **PHP**, which deploys the app to PHP App Server instances.

   Data source type

   An associated database server. For now, select **None**; we'll introduce database servers in Step 3: Add a Back-end Data Store.

   Repository type

   The app's repository type. The example app is stored in a **Git** repository.

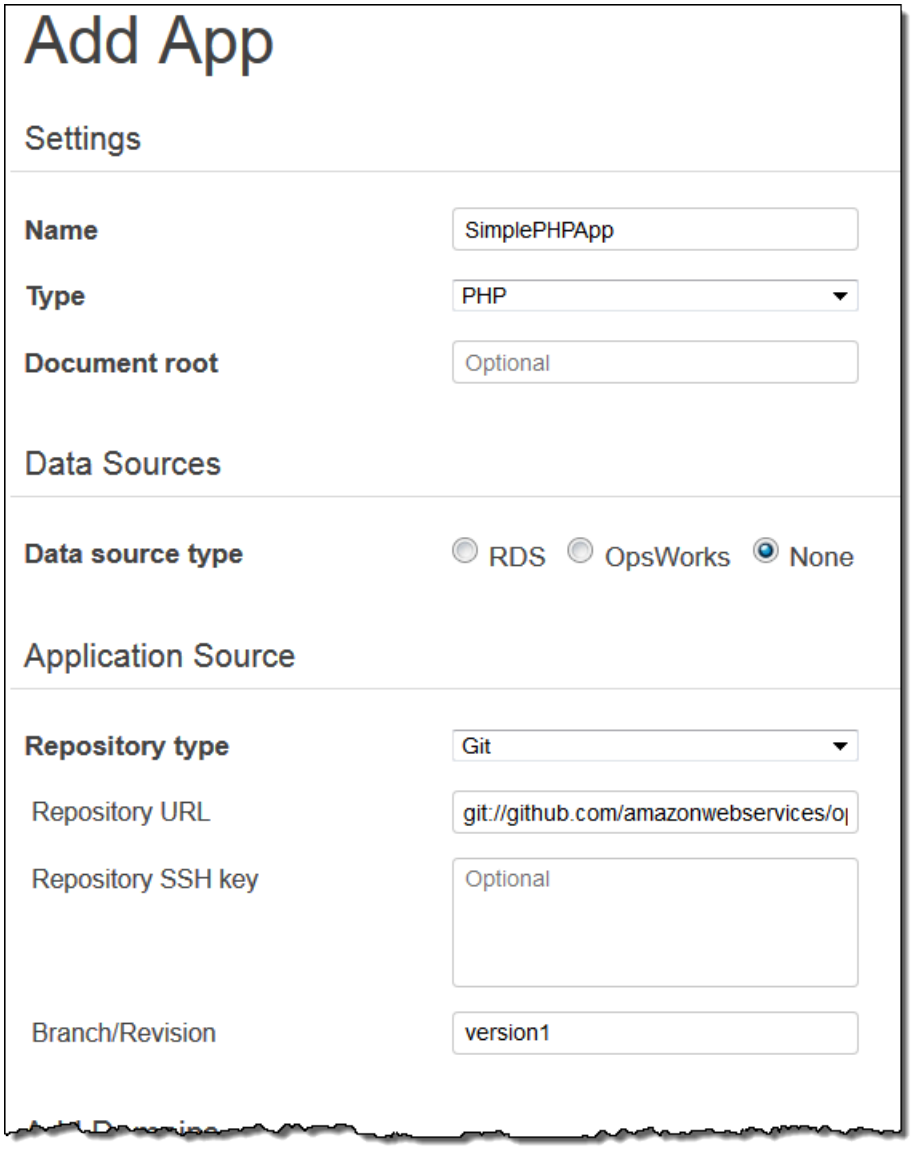
Repository URL

The app's repository URL. The example URL is: **`git://github.com/awslabs/opsworks-demo-php-simple-app.git`**

Branch/Revision

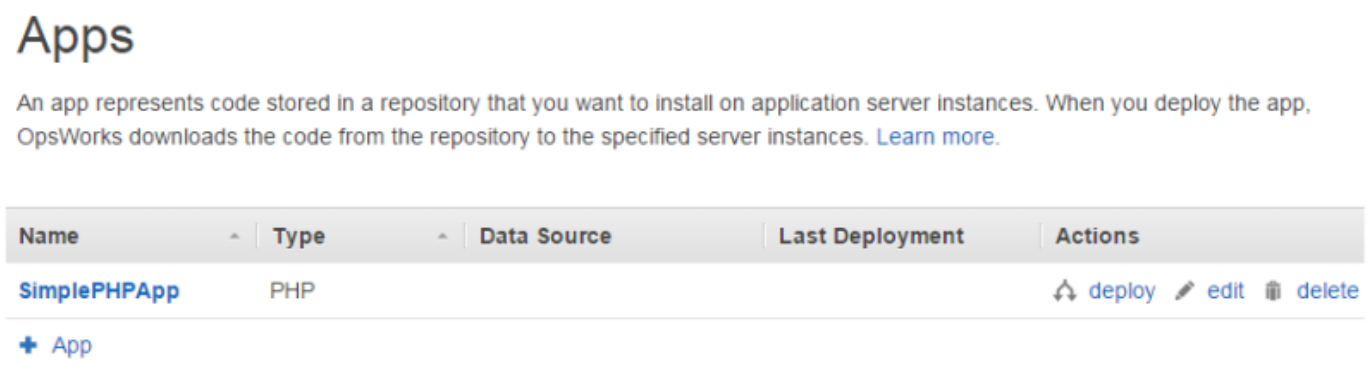
The app's branch or version. This part of the walkthrough uses the **`version1`** branch.

Keep the default values for the remaining settings and click **Add App**. For more information, see Adding Apps.



3. **Open the Deployment Page**

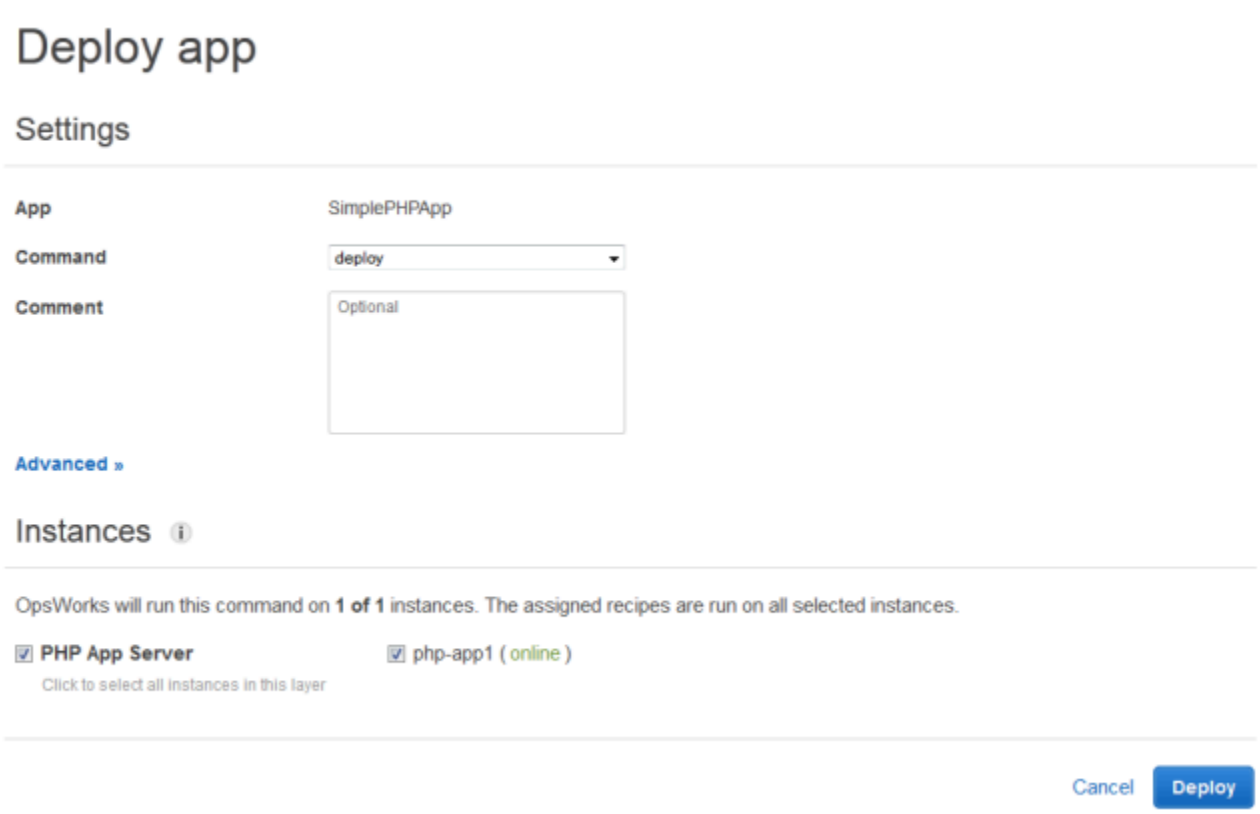
   To install the code on the server, you must *deploy* the app. To do so, click **deploy** in the SimplePHPApp **Actions** column.



4. **Deploy the App**

   When you deploy an app, the agent runs the Deploy recipes on the PHP App Server instance, which download and configure the application.

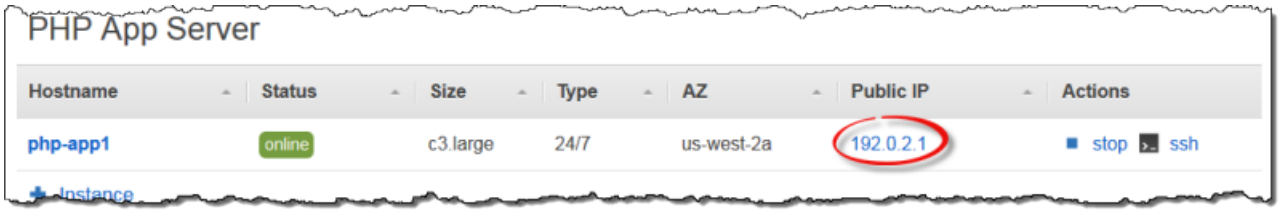
   Command should already be set to **deploy**. Keep the defaults for the other settings and click **Deploy** to deploy the app.



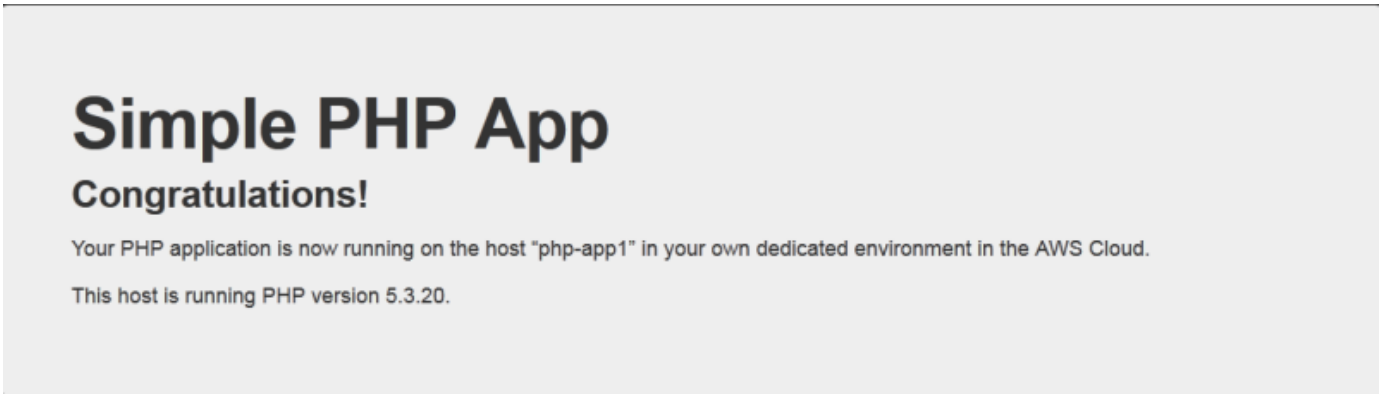
When deployment is complete, the **Deployment** page displays a **Status** of **Successful**, and **php-app1** will have a green check mark next to it.

5. **Run SimplePHPApp**

   SimplePHPApp is now installed and ready to go. To run it, click **Instances** in the navigation pane to go to the **Instances** page. Then click the php-app1 instance's public IP address.



   You should see a page such as the following in your browser.



Note

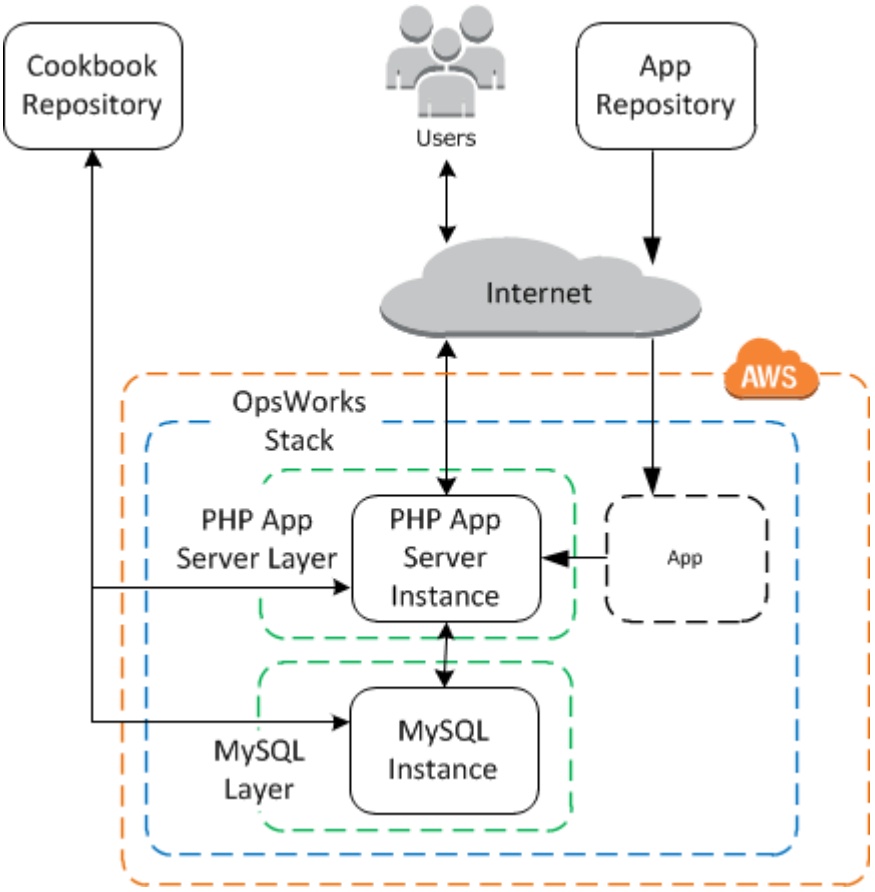
This walkthrough assumes that you will go on to the next section and ultimately complete the entire walkthrough in one session. If you prefer, you can stop at any point and continue later by signing in to OpsWorks Stacks and opening the stack. However, you are charged for any AWS resources that you use, such as online instances. To avoid unnecessary charges, you can stop your instance, which terminates the corresponding EC2 instance. You can start the instances again when you are ready to continue.

Step 3: Add a Back-end Data Store

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Step 2.1: Create a Stack - Chef 11 showed you how to create a stack that served a PHP application. However, that was a very simple application that did little more than display some static text. Production applications commonly use a back-end data store, yielding a stack configuration something like the illustration that follows.



This section shows how to extend MyStack to include a back-end MySQL database server. You need to do more than just add a MySQL server to the stack, though. You also have to configure the app to communicate properly with the database server. OpsWorks Stacks doesn't do this for you; you will need to implement some custom recipes to handle that task.

Topics

- Step 3.1: Add a Back-end Database
- Step 3.2: Update SimplePHPApp
- A Short Digression: Cookbooks, Recipes, and OpsWorks Stacks Attributes
- Step 3.3: Add the Custom Cookbooks to MyStack
- Step 3.4: Run the Recipes
- Step 3.5: Deploy SimplePHPApp, Version 2
- Step 3.6: Run SimplePHPApp

Step 3.1: Add a Back-end Database

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The new version of SimplePHPApp stores its data in a back-end database. OpsWorks Stacks supports two types of database servers:

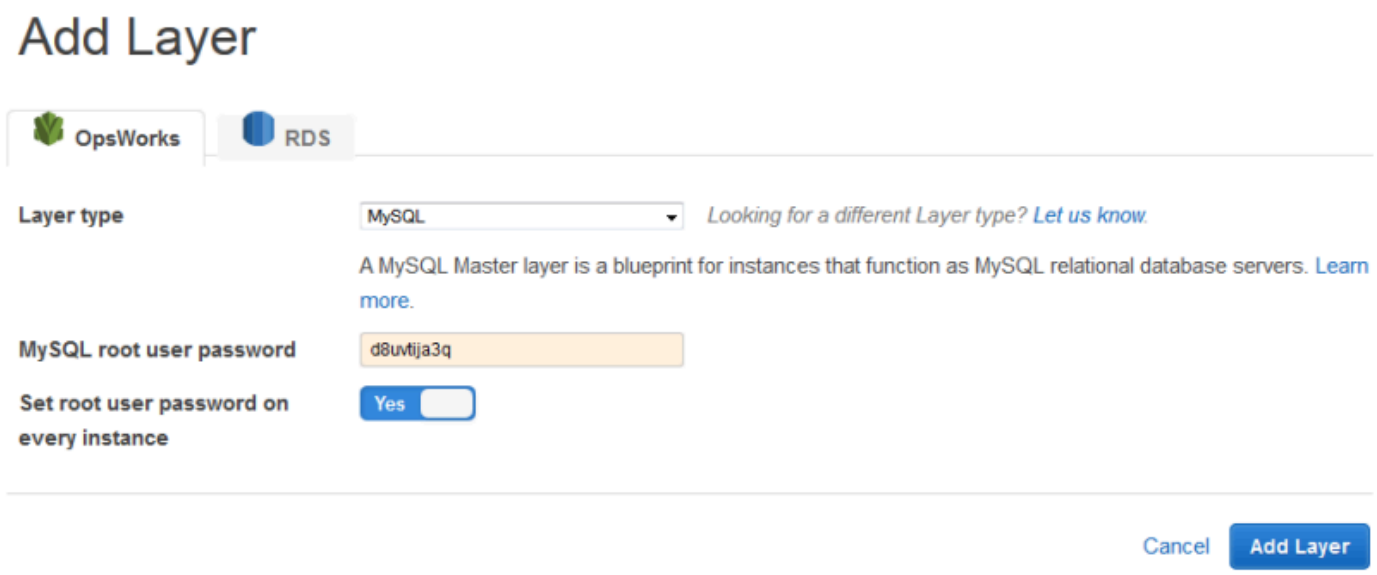
- The MySQL OpsWorks Stacks layer is a blueprint for creating Amazon EC2 instances that host a MySQL database master.
- The Amazon RDS service layer provides a way to incorporate an Amazon RDS instance into a stack.

You can also use other databases, such as Amazon DynamoDB, or create a custom layer to support databases such as MongoDB. For more information, see the section called "Using a Back-end Data Store".

This example uses a MySQL layer.

To add a MySQL layer to MyStack

1. On the **Layers** page, click **+ Layer**.
2. On the **Add Layer** page, for **Layer type**, select **MySQL**, accept the default settings, and click **Add Layer**.



To add an instance to the MySQL layer

1. On the **Layers** page's **MySQL** row, click **Add an instance**.
2. On the **Instances** page, under **MySQL**, click **Add an instance**.
3. Accept the defaults and click **Add instance**, but don't start it yet.

> **Note**
>
> OpsWorks Stacks automatically creates a database named using the app's short name, simplephpapp for this example. You'll need this name if you want to use Chef recipes to interact with the database.

Step 3.2: Update SimplePHPApp

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

To start, you need a new version of SimplePHPApp that uses a back-end data store. With OpsWorks Stacks, it's easy to update an application. If you use a Git or Subversion repository, you can have a separate repository branch for each app version. The example app stores a version of the app that uses a back-end database in the Git repository's version2 branch. You just need to update the app's configuration to specify the new branch and redeploy the app.

To update SimplePHPApp

1. **Open the App's Edit Page**

   In the navigation pane, click **Apps** and then click **edit** in the **SimplePHPApp** row's **Actions** column.

2. **Update the App's Configuration**

   Change the following settings.

   Branch/Revision

   This setting indicates the app's repository branch. The first version of SimplePHPApp didn't connect to a database. To use a the database-enabled version of the app, set this value to **`version2`**.

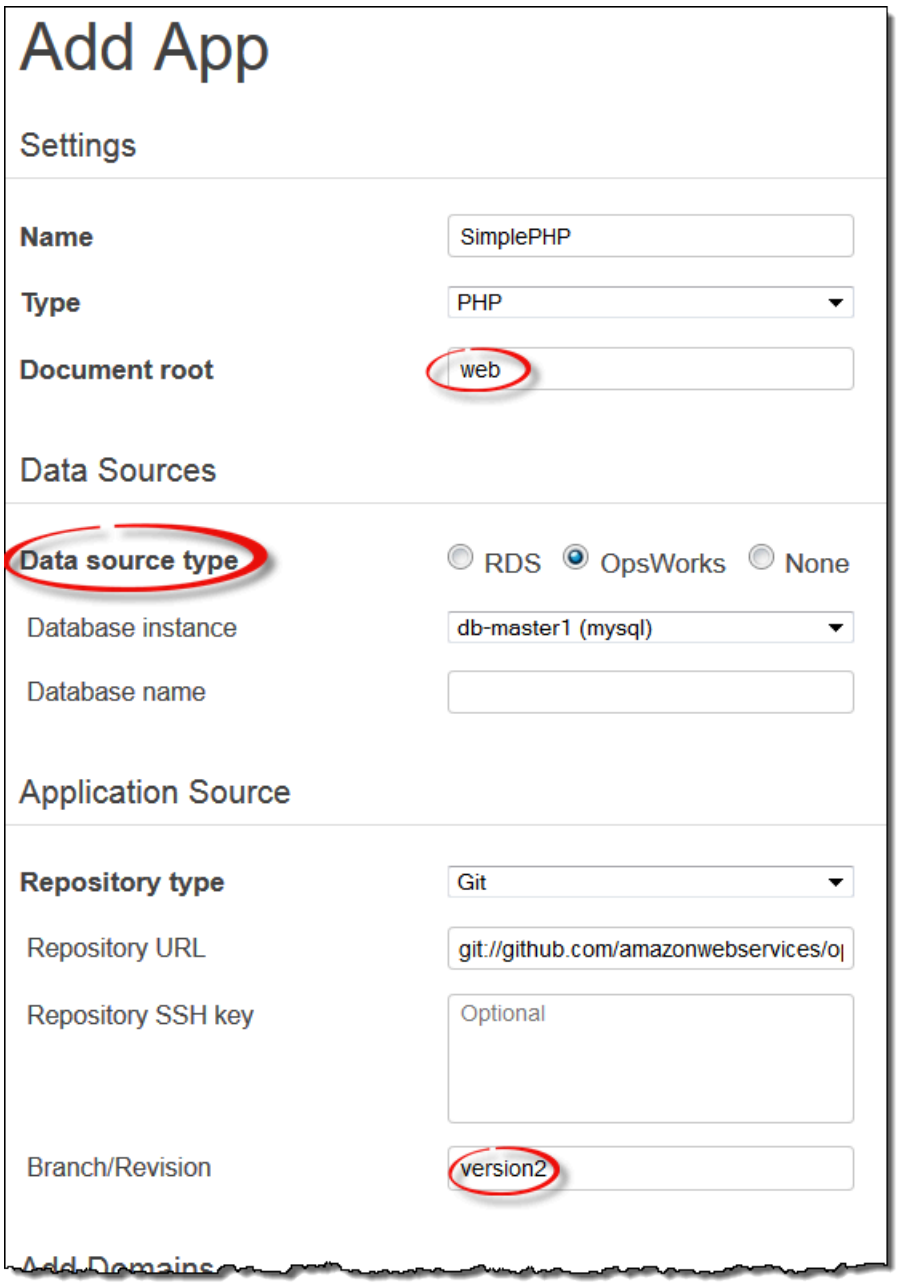
   Document root

   This setting specifies your app's root folder. The first version of SimplePHPApp used the default setting, which installs `index.php` in the server's standard root folder (`/srv/www` for PHP apps). If you specify a subfolder here—just the name, no leading '/'—OpsWorks Stacks appends it to the standard folder path. Version 2 of SimplePHPApp should go in `/srv/www/web`, so set **Document root** to **`web`**.

   Data source type

   This setting associates a database server with the app. The example uses the MySQL instance that you created in the previous step, so set **Data source type** to OpsWorks and **Database instance** to the instance you created in the previous step, **db-master1 (mysql)**. Leave **Database name** empty; OpsWorks Stacks will create a database on the server named with the app's short name, simplephpapp.

   Then click **Save** to save the new configuration.



3. Start the MySQL instance.

After you update an app, OpsWorks Stacks automatically deploys the new app version to any new app server instances when you start them. However, OpsWorks Stacks does not automatically deploy the new app version to existing server instances; you must do that manually, as described in Step 2.4: Create and Deploy an App - Chef 11. You could deploy the updated SimplePHPApp now, but for this example, it's better to wait a bit.

A Short Digression: Cookbooks, Recipes, and OpsWorks Stacks Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You now have app and database servers, but they aren't quite ready to use. You still need to set up the database and configure the app's connection settings. OpsWorks Stacks doesn't handle these tasks automatically, but it does support Chef cookbooks, recipes, and dynamic attributes. You can implement a pair of recipes, one to set up the database and one to configure the app's connection settings, and have OpsWorks Stacks run them for you.

The phpapp cookbook, which contains the required recipes, is already implemented and ready for use; you can just skip to Step 3.3: Add the Custom Cookbooks to MyStack if you prefer. If you'd like to know more, this section provides some background on cookbooks and recipes and describes how the recipes work. To see the cookbook itself, go to the phpapp cookbook.

Topics

- Recipes and Attributes
- Set Up the Database
- Connect the Application to the Database

Recipes and Attributes

A Chef recipe is basically a specialized Ruby application that performs tasks on an instance such as installing packages, creating configuration files, executing shell commands, and so on. Groups of related recipes are organized into *cookbooks*, which also contain supporting files such as templates for creating configuration files.

OpsWorks Stacks has a set of cookbooks that support the built-in layers. You can also create custom cookbooks with your own recipes to perform custom tasks on your instances. This topic provides a brief introduction to recipes and shows how to use them to set up the database and

configure the app's connection settings. For more information on cookbooks and recipes, see Cookbooks and Recipes or Customizing OpsWorks Stacks.

Recipes usually depend on Chef *attributes* for input data:

- Some of these attributes are defined by Chef and provide basic information about the instance such as the operating system.
- OpsWorks Stacks defines a set of attributes that contain information about the stack—such as the layer configurations—and about deployed apps—such as the app repository.

  You can add custom attributes to this set by assigning custom JSON to the stack or deployment.
- Your cookbooks can also define attributes, which are specific to the cookbook.

  The phpapp cookbook attributes are defined in `attributes/default.rb`.

For a complete list of OpsWorks Stacks attributes, see Stack Configuration and Deployment Attributes: Linux and Built-in Cookbook Attributes. For more information, see Overriding Attributes.

Attributes are organized in a hierarchical structure, which can be represented as a JSON object.

You incorporate this data into your application by using Chef node syntax, like the following:

```
[:deploy][:simplephpapp][:database][:username]
```

The `deploy` node has a single app node, `simplephpapp`, that contains information about the app's database, Git repository, and so on. The example represents the value of the database user name, which resolves to `root`.

Set Up the Database

The MySQL layer's built-in Setup recipes automatically create a database for the app named with the app's shortname, so for this example you already have a database named simplephpapp. However, you need to finish the setup by creating a table for the app to store its data. You could create the table manually, but a better approach is to implement a custom recipe to handle the task, and have OpsWorks Stacks run it for you. This section describes how the recipe, `dbsetup.rb`, is implemented. The procedure for having OpsWorks Stacks run the recipe is described later.

To see the recipe in the repository, go to dbsetup.rb. The following example shows the `dbsetup.rb` code.

`execute` is a *Chef resource* that executes a specified command. In this case, it's a MySQL command that creates a table. The `not_if` directive ensures that the command does not run if the specified table already exists. For more information on Chef resources, see About Resources and Providers.

The recipe inserts attribute values into the command string, using the node syntax discussed earlier. For example, the following inserts the database's user name.

```
#{deploy[:database][:username]}
```

Let's unpack this somewhat cryptic code:

- For each iteration, `deploy` is set to the current app node, so it resolves to `[:deploy][:app_name]`. For this example, it resolves to `[:deploy][:simplephpapp]`.
- Using the deployment attribute values shown earlier, the entire node resolves to `root`.
- You wrap the node in #{ } to insert it into a string.

Most of the other nodes resolve in a similar way. The exception is `#{node[:phpapp][:dbtable]}`, which is defined by the custom cookbook's attributes file and resolves to the table name, `urler`. The actual command that runs on the MySQL instance is therefore:

```
"/usr/bin/mysql
    -uroot
    -pvjud1hw5v8
    simplephpapp
    -e'CREATE TABLE urler(
       id INT UNSIGNED NOT NULL AUTO_INCREMENT,
       author VARCHAR(63) NOT NULL,
       message TEXT,
       PRIMARY KEY (id))'
"
```

This command creates a table named `urler` with id, author, and message fields, using the credentials and database name from the deployment attributes.

Connect the Application to the Database

The second piece of the puzzle is the application, which needs connection information such as the database password to access the table. SimplePHPApp effectively has only one working file, `app.php`; all `index.php` does is load `app.php`.

app.php includes db-connect.php, which handles the database connection, but that file is not in the repository. You can't create db-connect.php in advance because it defines the database based on the particular instance. Instead, the appsetup.rb recipe generates db-connect.php using connection data from the deployment attributes.

To see the recipe in the repository, go to appsetup.rb. The following example shows the appsetup.rb code.

Like dbsetup.rb, appsetup.rb iterates over apps in the deploy node—just simplephpapp again—. It runs a code block with a script resource and a template resource.

The script resource installs Composer—a dependency manager for PHP applications. It then runs Composer's install command to install the dependencies for the sample application to the app's root directory.

The template resource generates db-connect.php and puts it in /srv/www/simplephpapp/current. Note the following:

- The recipe uses a conditional statement to specify the file owner, which depends on the instance's operating system.
- The only_if directive tells Chef to generate the template only if the specified directory exists.

A template resource operates on a template that has essentially the same content and structure as the associated file but includes placeholders for various data values. The source parameter specifies the template, db-connect.php.erb, which is in the phpapp cookbook's templates/default directory, and contains the following:

When Chef processes the template, it replaces the <%= => placeholders with the value of the corresponding variables in the template resource, which are in turn drawn from the deployment attributes. The generated file is therefore:

Step 3.3: Add the Custom Cookbooks to MyStack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You store custom cookbooks in a repository, much like apps. Each stack can have a repository that contains one set of custom cookbooks. You then direct OpsWorks Stacks to install your custom cookbooks on the stack's instances.

1. Click **Stack** in the navigation pane to see the page for the current stack.
2. Click **Stack Settings**, and then click **Edit**.
3. Modify the stack configuration as follows:
   - **Use custom Chef Cookbooks** – **Yes**
   - **Repository type** – **Git**
   - **Repository URL** – **`git://github.com/amazonwebservices/opsworks-example-cookbooks.git`**
4. Click **Save** to update the stack configuration.



OpsWorks Stacks then installs the contents of your cookbook repository on all of the stack's instances. If you create new instances, OpsWorks Stacks automatically installs the cookbook repository.

Note

If you need to update any of your cookbooks, or add new cookbooks to the repository, you can do so without touching the stack settings. OpsWorks Stacks will automatically install the updated cookbooks on all new instances. However, OpsWorks Stacks does not automatically install updated cookbooks on the stack's online instances. You must explicitly

direct OpsWorks Stacks to update the cookbooks by running the `Update Cookbooks` stack command. For more information, see Run Stack Commands.

Step 3.4: Run the Recipes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

After you have your custom cookbook, you need to run the recipes on the appropriate instances. You could run them manually. However, recipes typically need to be run at predictable points in an instance's lifecycle, such as after the instance boots or when you deploy an app. This section describes a much simpler approach: have OpsWorks Stacks automatically run them for you at the appropriate time.

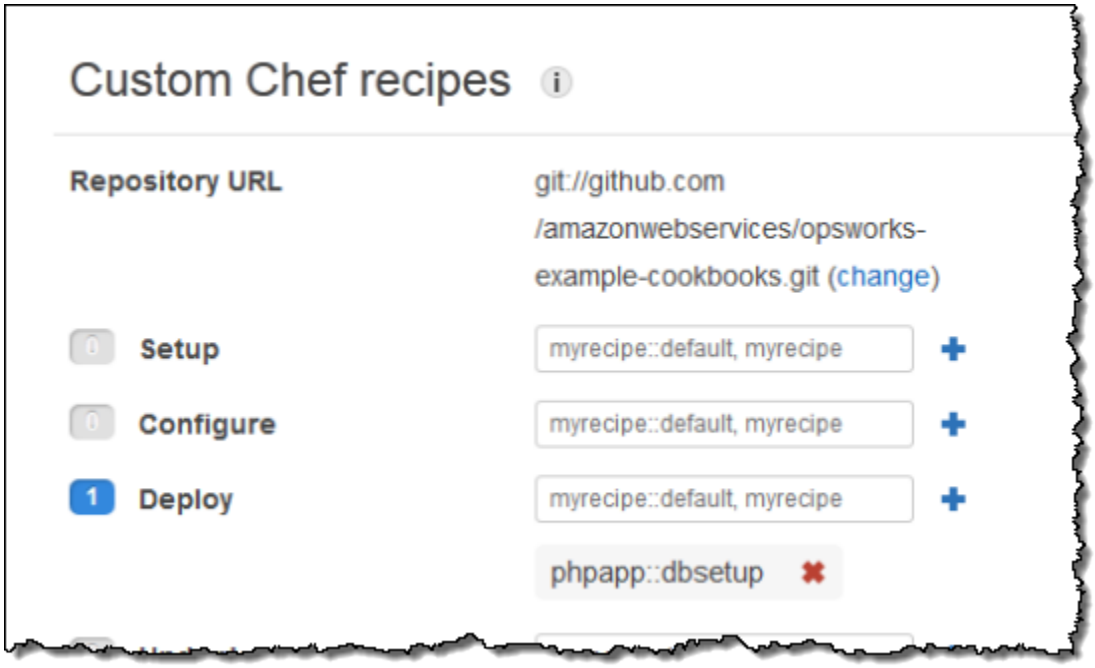
OpsWorks Stacks supports a set of lifecycle events that simplify running recipes. For example, the Setup event occurs after an instance finishes booting and the Deploy event occurs when you deploy an app. Each layer has a set of built-in recipes associated with each lifecycle event. When a lifecycle event occurs on an instance, the agent runs the associated recipes for each of the instance's layers. To have OpsWorks Stacks run a custom recipe automatically, add it to the appropriate lifecycle event on the appropriate layer and the agent will run the recipe after the built-in recipes are finished.

For this example, you need to run two recipes, `dbsetup.rb` on the MySQLinstance and `appsetup.rb` on the PHP App Server instance.

> **Note**
>
> You specify recipes on the console by using the *`cookbook_name`*::*`recipe_name`* format, where *`recipe_name`* does not include the .rb extension. For example, you refer to `dbsetup.rb` as **`phpapp::dbsetup`**.

To assign custom recipes to lifecycle events

1. On the **Layers** page, for MySQL, click **Recipes** and then click **Edit**.
2. In the **Custom Chef recipes** section, enter **`phpapp::dbsetup`** for **Deploy**.



3. Click the **+** icon to assign the recipe to the event and click **Save** to save the new layer configuration.
4. Return to the **Layers** page and repeat the procedure to assign **`phpapp::appsetup`** to the **PHP App Server** layer's **Deploy** event.

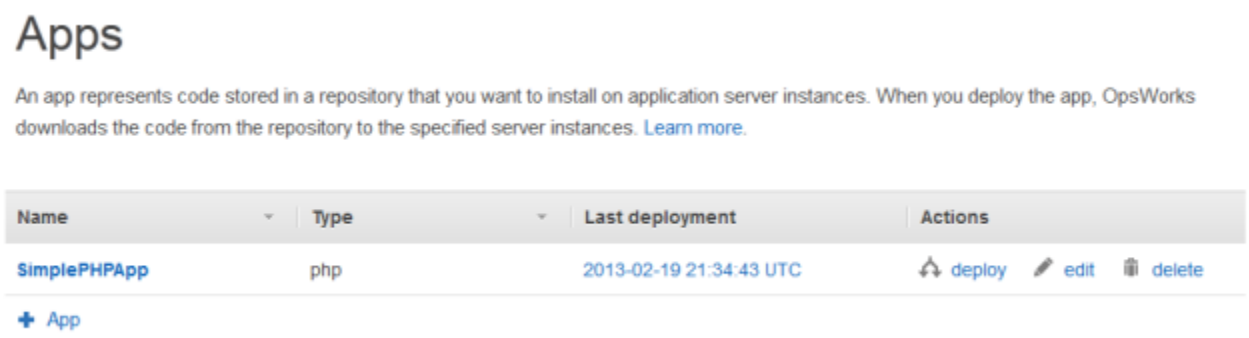
Step 3.5: Deploy SimplePHPApp, Version 2

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

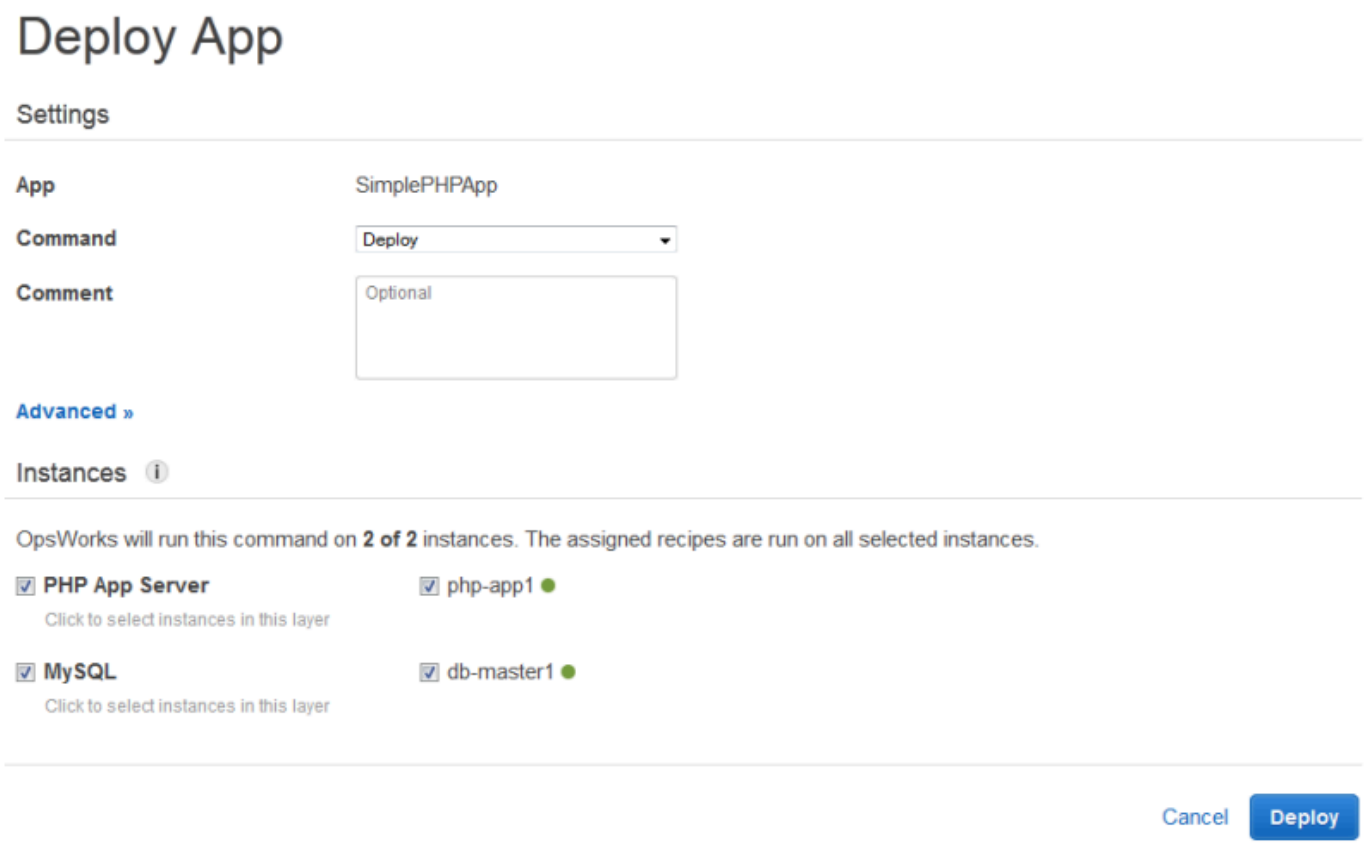
The final step is to deploy the new version of SimplePHPApp.

To deploy SimplePHPApp

1. On the **Apps** page, click **deploy** in the **SimplePHPApp** app's **Actions**.



2. Accept the defaults and click **Deploy**.



When you click **Deploy** on the **Deploy App** page, you trigger a Deploy lifecycle event, which notifies the agents to run their Deploy recipes. By default, you trigger the event on all of the stack's instances. The built-in Deploy recipes deploy the app only to the appropriate instances for the app type, PHP App Server instances in this case. However, it is often useful to trigger

the Deploy event on other instances, to allow them to respond to the app deployment. In this case, you also want to trigger Deploy on the MySQL instance to set up the database.

Note the following:

- The agent on the PHP App Server instance runs the layer's built-in recipe, followed by `appsetup.rb`, which configures the app's database connection.
- The agent on the MySQL instance doesn't install anything, but it runs `dbsetup.rb` to create the urler table.

When the deployment is complete, the **Status** will change to **successful** on the **Deployment** page.

Step 3.6: Run SimplePHPApp

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

After the deployment status changes to **successful**, you can run the new SimplePHPApp version, as follows.

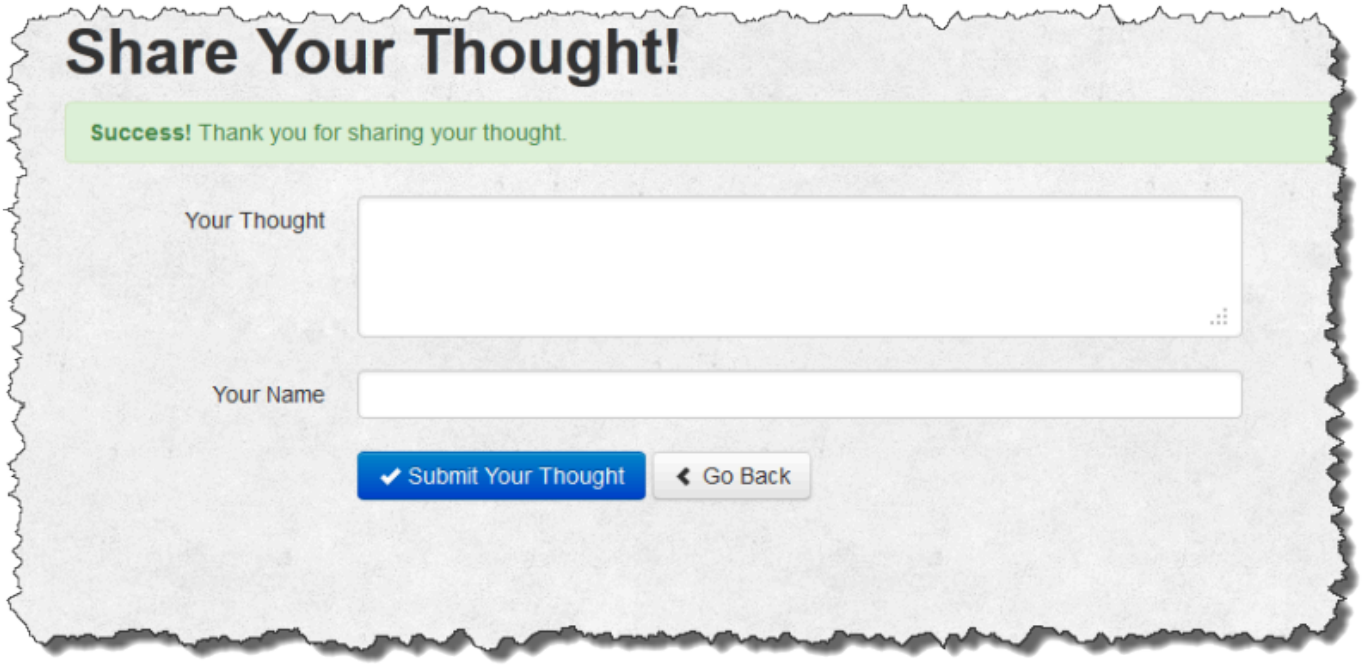
To run SimplePHPApp

1. On the **Instances** page, click the public IP address in the **php-app1** row.

   You should see the following page in your browser.



2. Click **Share Your Thought** and type something like **`Hello world!`** for **Your Thought** and your name for **Your Name**. Then click the **Submit Your Thought** to add the message to the database.

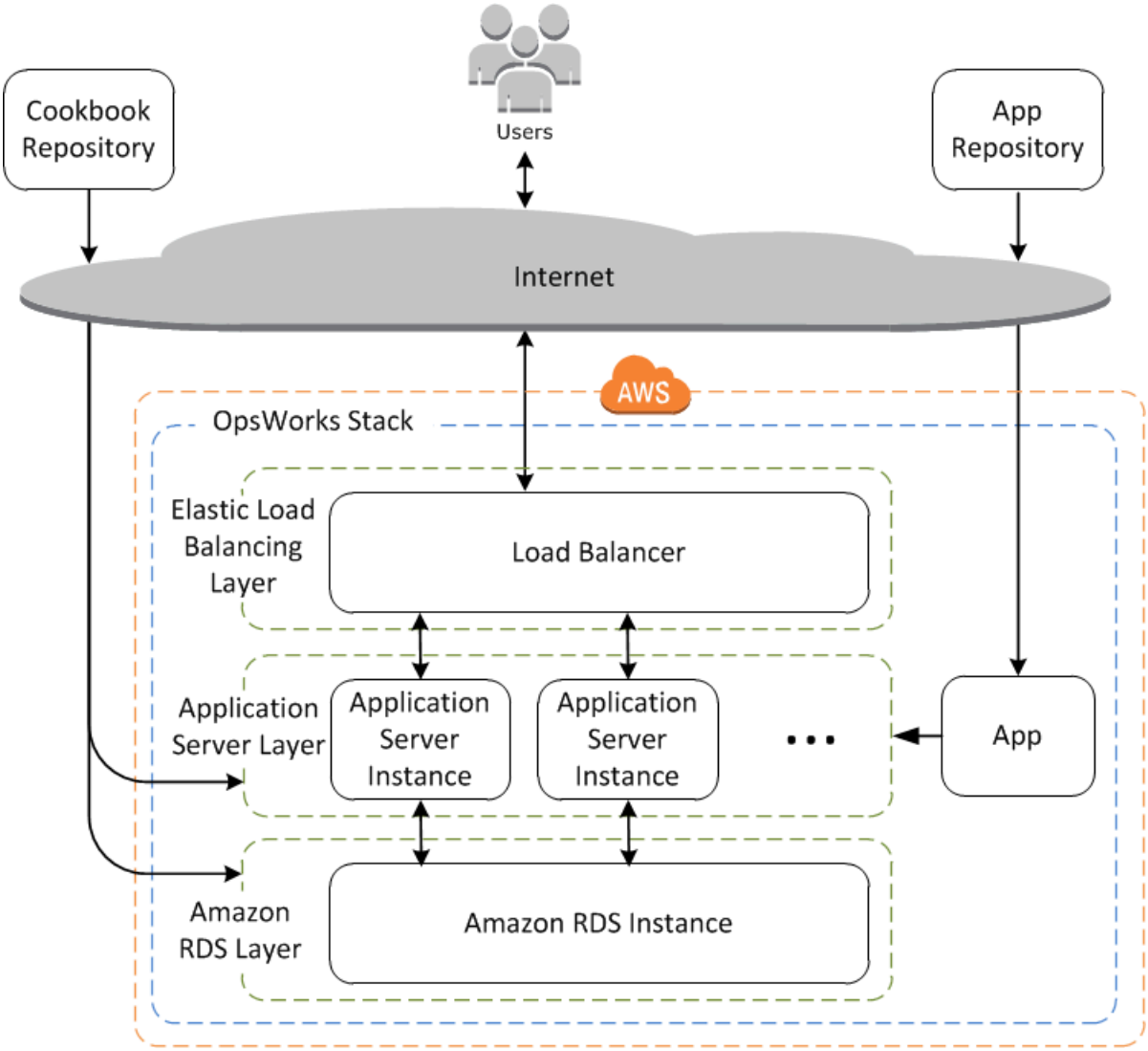


3. Click **Go Back** to view all the messages in the database.

Step 4: Scale Out MyStack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

MyStack currently has only one application server. A production stack will probably need multiple application servers to handle the incoming traffic and a load balancer to distribute the incoming traffic evenly across the application servers. The architecture will look something like the following:



OpsWorks Stacks makes it easy to scale out stacks. This section describes the basics of how to scale out a stack by adding a second 24/7 PHP App Server instance to MyStack and putting both instances behind an Elastic Load Balancing load balancer. You can easily extend the procedure to add an arbitrary number of 24/7 instances, or you can use time-based or load-based instances to have OpsWorks Stacks scale your stack automatically. For more information, see Managing load with time-based and load-based instances.

Step 4.1: Add a Load Balancer

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Elastic Load Balancing is an AWS service that automatically distributes incoming application traffic across multiple Amazon EC2 instances. In addition to distributing traffic, Elastic Load Balancing does the following:

- Detects unhealthy Amazon EC2 instances.

  It reroutes traffic to the remaining healthy instances until the unhealthy instances have been restored.
- Automatically scales request handling capacity in response to incoming traffic

Note

A load balancer can serve two purposes. The obvious one is to equalize the load on your application servers. In addition, many sites prefer to isolate their application servers and databases from direct user access. With OpsWorks Stacks, you can do this by running your stack in a virtual private cloud (VPC) with a public and private subnet, as follows.

- Put the application servers and database in the private subnet, where they can be accessed by other instances in the VPC but not by users.

- Direct user traffic to a load balancer in the public subnet, which then forwards the traffic to the application servers in the private subnet and returns responses to users.

For more information, see Running a Stack in a VPC. For an AWS CloudFormation template that extends the example in this walkthrough to run in a VPC, download the `OpsWorksVPCtemplates.zip` file.

Although Elastic Load Balancing is often referred to as a layer, it works a bit differently than the other built-in layers. Instead of creating a layer and adding instances to it, you create an Elastic Load Balancing load balancer by using the Amazon EC2 console and then attach it to one of your existing layers, usually an application server layer. OpsWorks Stacks then registers the layer's existing instances with the service and automatically adds any new instances. The following procedure describes how to add a load balancer to MyStack's PHP App Server layer.

Note

OpsWorks Stacks does not support Application Load Balancer. You can only use Classic Load Balancer with OpsWorks Stacks.

To attach a load balancer to the PHP App Server layer

1. Use the Amazon EC2 console to create a new load balancer for MyStack. The details depend on whether your account supports EC2 Classic. For more information, see Getting Started with Elastic Load Balancing. When you run the **Create Load Balancer** wizard, configure the load balancer as follows:

   Define Load Balancer

   Assign the load balancer an easily recognizable name, like PHP-LB, to make it easier to locate in the OpsWorks Stacks console. Then choose **Continue** to accept defaults for the remaining settings.

   If you choose a VPC with one or more subnets from the **Create LB Inside** menu, you must select a subnet for each availability zone where you want traffic to be routed by your load balancer.

Assign Security Groups

If your account supports default VPC, the wizard displays this page to determine the load balancer's security group. It does not display this page for EC2 Classic.

For this walkthrough, choose **default VPC security group**.

Configure Security Settings

If you chose **HTTPS** as the **Load Balancer Protocol** on the **Define Load Balancer** page, configure certificate, cipher, and SSL protocol settings on this page. For this walkthrough, accept defaults, and choose **Configure Health Check**.

Configure Health Check

Set the ping path to **/** and accept defaults for remaining settings.

Add EC2 Instances

Choose **Continue**; OpsWorks Stacks automatically registers instances with the load balancer.

Add Tags

Add tags to help you find . Each tag is a key and value pair; for example, you could specify **`Description`** as the key and **`Test LB`** as the value for the purposes of the walkthrough.

Review

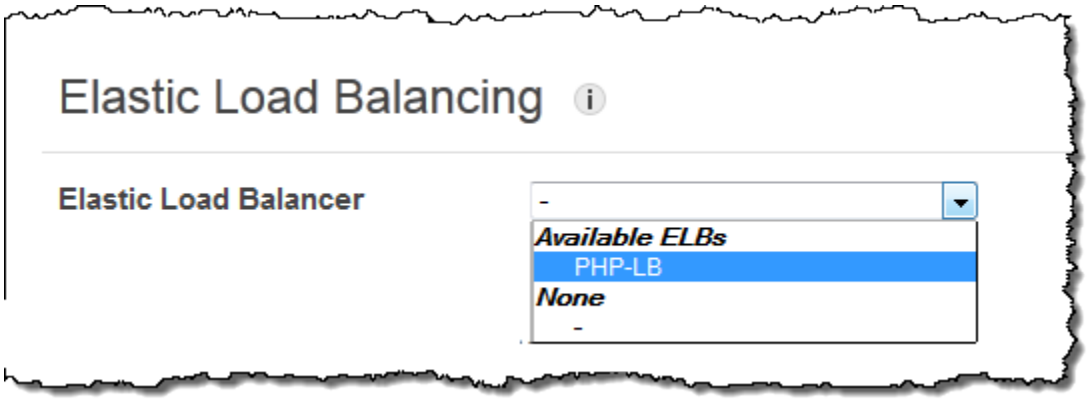
Review your choices, choose **Create**, and then choose **Close**, which starts the load balancer.

2. If your account supports default VPC, after you start the load balancer, you must ensure that its security group has appropriate ingress rules. The default rule does not accept any inbound traffic.

   1. Choose **Security Groups** in the Amazon EC2 navigation pane.
   2. Select **default VPC security group**
   3. Choose **Edit** on the **Inbound** tab.
   4. For this walkthrough, set **Source** to **Anywhere**, which directs the load balancer to accept incoming traffic from any IP address.

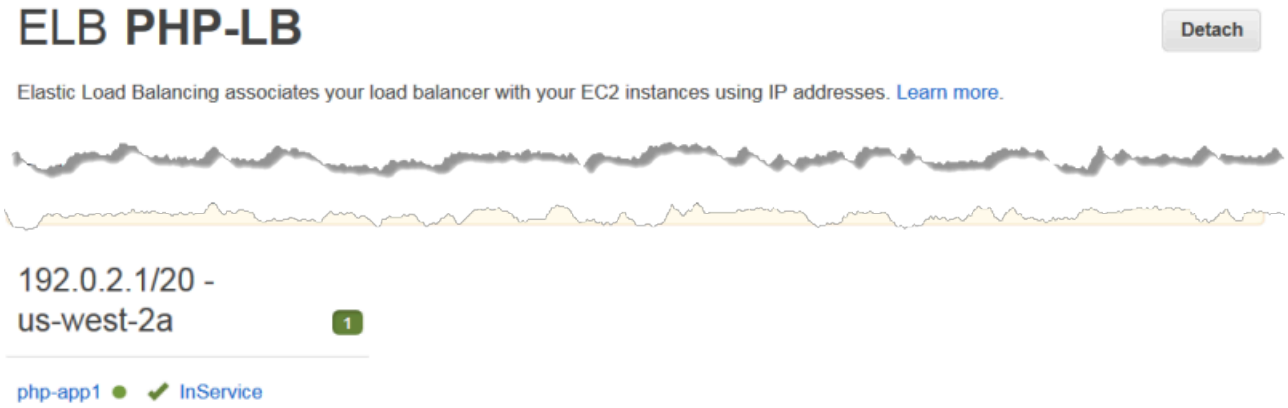
3. Return to the OpsWorks Stacks console. On the **Layers** page, choose the layer's **Network** link, and then choose **Edit**.

4. Under **Elastic Load Balancing**, choose the load balancer that you created in Step 1, and then choose **Save**.



   After you have attached the load balancer to the layer, OpsWorks Stacks automatically registers the layer's current instances, and adds new instances as they come online.

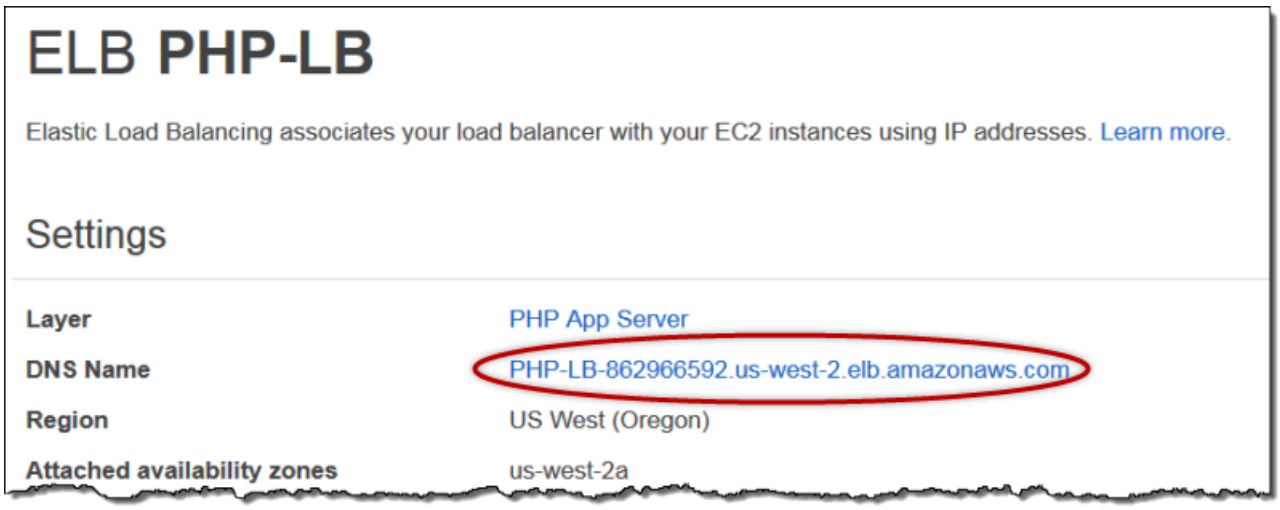
5. On the **Layers** page, click the load balancer's name to open its details page. When registration is complete and the instance passes a health check, OpsWorks Stacks shows a green check mark next to the instance on the load balancer page.



You can now run SimplePHPApp by sending a request to the load balancer.

To run SimplePHPApp through the load balancer

1. Open load balancer's details page again, if it is not already open.
2. On the properties page, verify the instance's health-check status and click the load balancer's DNS name to run SimplePHPApp. The load balancer forwards the request to the PHP App Server instance and returns the response, which should look exactly the same as the response you get when you click the PHP App Server instance's public IP address.



Note

OpsWorks Stacks also supports the HAProxy load balancer, which might have advantages for some applications. For more information, see HAProxy OpsWorks Stacks Layer.

Step 4.2: Add PHP App Server Instances

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Now the load balancer is in place, you can scale out the stack by adding more instances to the PHP App Server layer. From your perspective, the operation is seamless. Each time a new PHP App Server instance comes online, OpsWorks Stacks automatically registers it with the load balancer and deploys SimplePHPApp, so the server can immediately start handling incoming traffic. For brevity, this topic shows how to add one additional PHP App Server instance, but you can use the same approach to add as many as you need.

To add another instance to the PHP App Server layer

1. On the Instances page, click **+ Instance** under **PHP App Server**.
2. Accept the default settings and click **Add Instance**.
3. Click **start** to start the instance.

Step 4.3: Monitor MyStack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

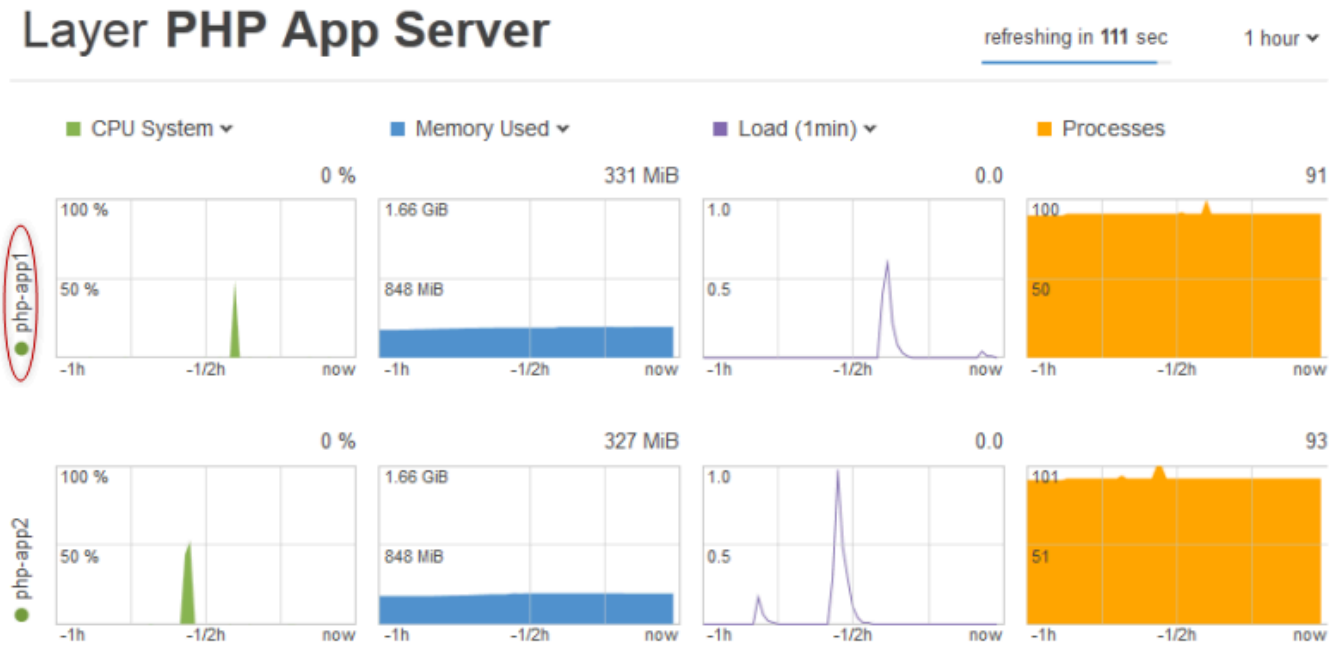
OpsWorks Stacks uses Amazon CloudWatch to provide metrics for a stack and summarizes them for your convenience on the **Monitoring** page. You can view metrics for the entire stack, a specified layer, or a specified instance.

To monitor MyStack

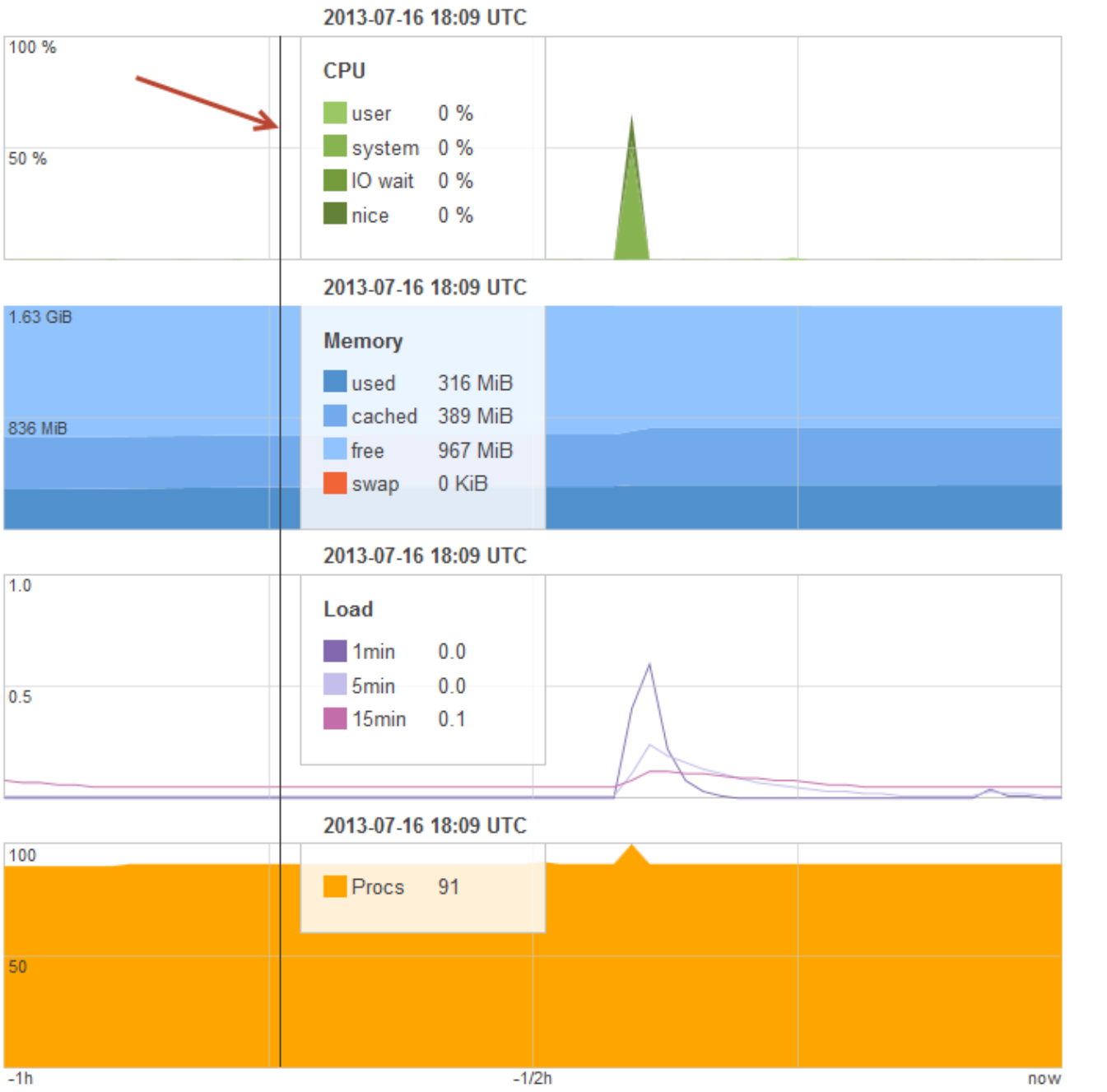
1. In the navigation pane, click **Monitoring**, which displays a set of graphs with average metrics for each layer. You can use the menus for **CPU System**, **Memory Used**, and **Load** to display different related metrics.



2. Click **PHP App Server** to see metrics for each of the layer's instances.



3. Click **php-app1** to see metrics for that instance. You can see metrics for any particular point in time by moving the slider.


Instance php-app1
refreshing in
2013-07-16 18:09 UTC
100 %
50 %
CPU
user 0 %
system 0 %
IO wait 0 %
nice 0 %
2013-07-16 18:09 UTC
1.63 GiB
836 MiB
Memory
used 316 MiB
cached 389 MiB
free 967 MiB
swap 0 KiB
2013-07-16 18:09 UTC
1.0
0.5
Load
1min 0.0
5min 0.0
15min 0.1
2013-07-16 18:09 UTC
100
50
Procs 91
-1h
-1/2h
now

> **Note**
>
> OpsWorks Stacks also supports the Ganglia monitoring server, which might have advantages for some applications. For more information, see Ganglia Layer.

Step 5: Delete MyStack

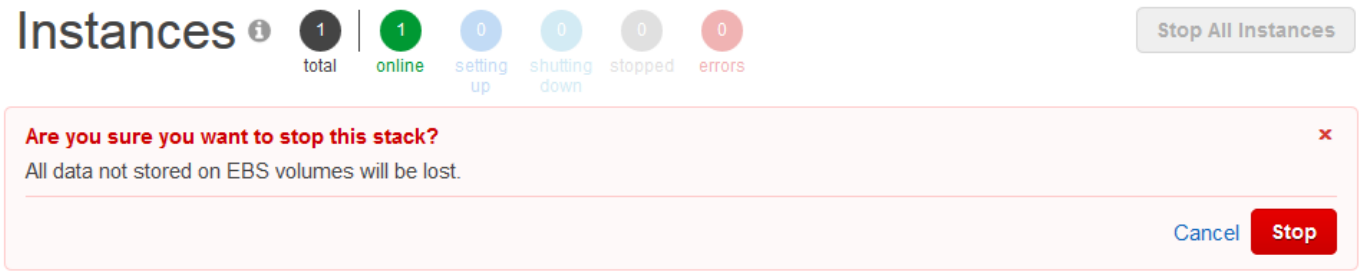
> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

As soon as you begin using AWS resources like Amazon EC2 instances you are charged based on your usage. If you are finished for now, you should stop the instances so that you do not incur any unwanted charges. If you don't need the stack anymore, you can delete it.

To delete MyStack

1. **Stop all Instances**

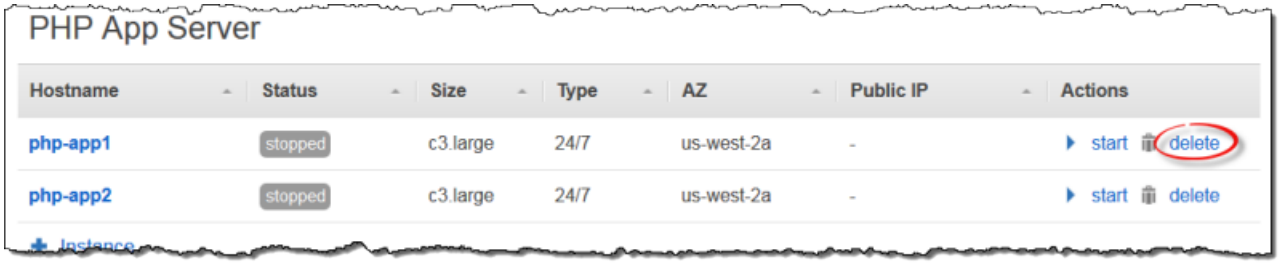
   On the **Instances** page, click **Stop All Instances** and click **Stop** when asked confirm the operation.



   After you click **Stop**, OpsWorks Stacks terminates the associated Amazon EC2 instances, but not any associated resources such as Elastic IP addresses or Amazon EBS volumes.

2. **Delete all Instances**

   Stopping the instance just terminates the associated Amazon EC2 instances. After the instances status is in the stopped stated, you must delete each instance. In the **PHP App Server** layer click **delete** in the php-app1 instance's **Actions** column.

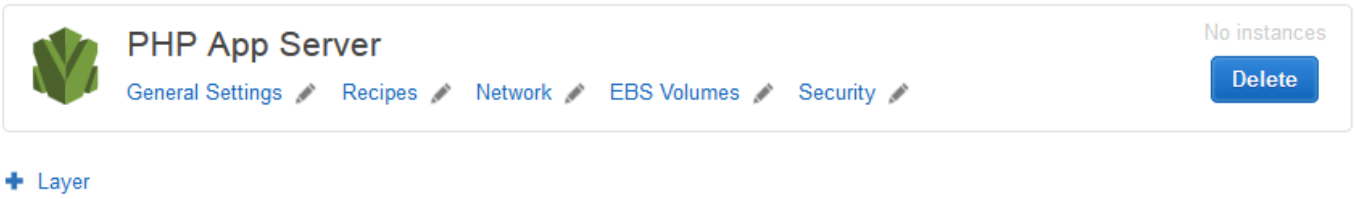


   OpsWorks Stacks then asks you to confirm the deletion, and shows you any dependent resources. You can choose to keep any or all of these resources. This example has no dependent resources, so just click **Delete**.

   Repeat the process for php-app2 and the **MySQL** instance, db-master1. Notice that db-master1 has an associated Amazon Elastic Block Store volume, which is selected by default. Leave it selected to delete the volume along with the instance.

3. **Delete the Layers.**

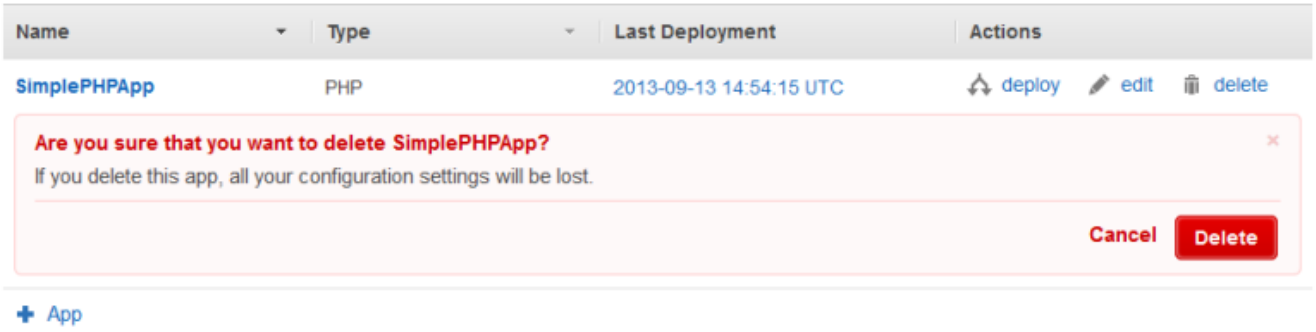
   On the **Layers** page, click **Delete** and then click **Delete** to confirm.



   Repeat the process for the **MySQL** layer.

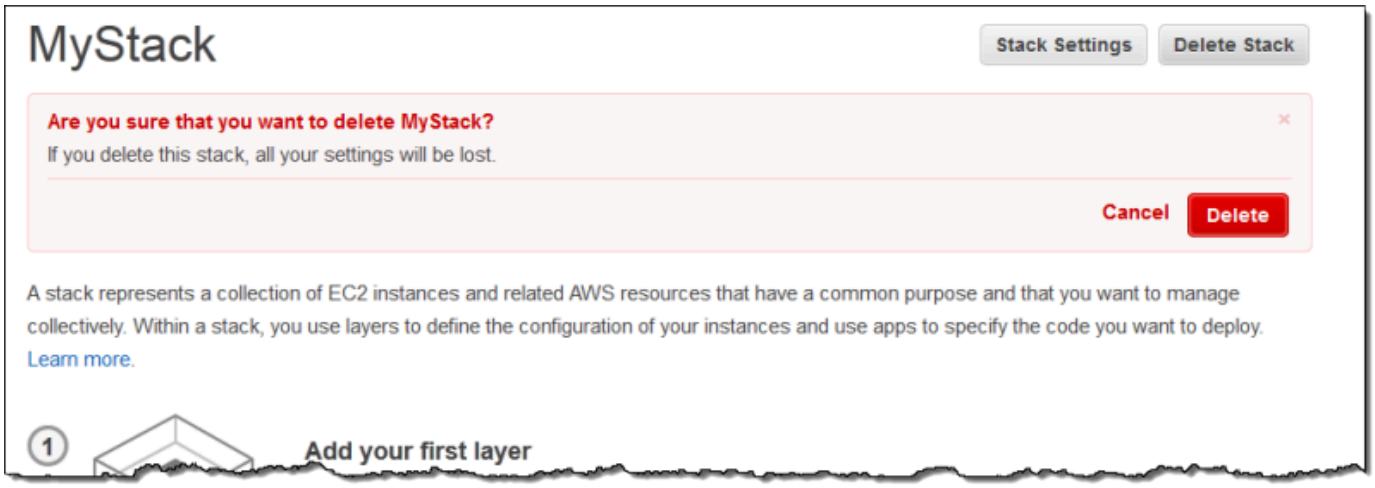
4. **Delete the App**

   On the **Apps** page, click **delete** in the **SimplePHPApp** app's **Actions** column, and then click **Delete** to confirm.



5. **Delete MyStack**

   On the **Stack** page, click **Delete Stack** and then click **Delete** to confirm.



You have now reached the end of this walkthrough.

Creating Your First Node.js Stack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This example describes how to create a Linux stack that supports a Node.js application server and how to deploy a simple application. The stack consists of the following components:

- A Node.js App Server layer with two instances
- An Elastic Load Balancing load balancer to distribute traffic to the application server instances
- An Amazon Relational Database Service (Amazon RDS) service layer that provides a backend database

Topics

- Prerequisites
- Implementing the Application
- Creating the Database Server and Load Balancer
- Creating the Stack
- Deploying the Application
- What Next?

Prerequisites

This walkthrough assumes the following:

- You have an AWS account and a basic understanding of how to use OpsWorks Stacks.

  If you are new to OpsWorks Stacks or to AWS, learn the basics by completing the introductory tutorial in Getting Started with Chef 11 Linux Stacks.
- You have a basic understanding of how to implement a Node.js application.

  If you are new to Node.js, learn the basics by completing an introductory tutorial, such as Node: Up and Running.
- You have already created at least one stack in the AWS region that you plan to use for this example.

  When you create the first stack in a region, OpsWorks Stacks creates an Amazon Elastic Compute Cloud (Amazon EC2) security group for each layer type. You need these security groups to create the Amazon RDS database (DB) instance. If you are new to OpsWorks Stacks, we recommend that you use the same region for this example that you did when you followed the tutorial in Getting Started with Chef 11 Linux Stacks. If you want to use a new region, create a new stack in the region; the stack does not need to have layers or instances. As soon as you create the stack, OpsWorks Stacks automatically adds a set of security groups to the region.

- You will create your stack in a default VPC.

  You can use EC2-Classic for this walkthrough, but some of the details will differ slightly. For example, with EC2-Classic, you specify an instance's Availability Zone (AZ) instead of its subnet.
- Your IAM user has full-access permissions for OpsWorks Stacks.

  For security reasons, we strongly recommend that you do not use your account's root credentials for this walkthrough. Instead, create a user with OpsWorks Stacks full-access permissions and use those credentials with OpsWorks Stacks. For more information, see Creating an Administrative User.

Implementing the Application

This walkthrough uses a simple Express application that connects to the Amazon RDS DB instance and lists the instance's databases.

To implement the application, create a directory named nodedb in a convenient location on your workstation and add the following three files to it.

Topics

- The Package Descriptor
- The Layout File
- The Code File

The Package Descriptor

To create the application's package descriptor, add a file named `package.json` with the following contents to the nodedb directory. `package.json` is required for Express applications and must be located in the application's root directory.

```
{
  "name": "Nodejs-DB",
  "description": "Node.js example application",
  "version": "0.0.1",
  "dependencies": {
    "express": "*",
    "ejs": "*",
```

```
    "mysql": "*"
  }
}
```

This `package.json` example is fairly minimal. It defines the required `name` and `version` attributes and lists the dependent packages:

- `express` references the Express package.
- `ejs` references the EJS package, which the application uses to insert text into an HTML layout file.
- `mysql` references the node-mysql package, which the application uses to connect to the RDS instance.

For more information on package descriptor files, see package.json.

The Layout File

To create the application's layout file, add a `views` directory to the nodedb directory, and then add a file to `views` named `index.html` with the following contents:

```
<!DOCTYPE html>
<html>
<head>
  <title>AWS Opsworks Node.js Example</title>
</head>
<body>
  <h1>AWS OpsWorks Node.js Example</h1>
    <p>Amazon RDS Endpoint: <i><%= hostname %></i></p>
    <p>User: <i><%= username %></i></p>
    <p>Password: <i><%= password %></i></p>
    <p>Port: <i><%= port %></i></p>
    <p>Database: <i><%= database %></i></p>

    <p>Connection: <%= connectionerror %></p>
    <p>Databases: <%= databases %></p>
</body>
</html>
```

For this example, the layout file is a simple HTML document that displays some data from Amazon RDS. Each `<%= ... =>` element represents the value of a variable that is defined in the application's code file, which we create next.

The Code File

To create the application's code file, add a `server.js` file to the nodedb directory with the following contents.

> **Important**
>
> With OpsWorks Stacks, a Node.js application's main code file must be named `server.js` and be located in the application's root folder.

```
var express = require('express');
var mysql = require('mysql');
var dbconfig = require('opsworks'); //[1] Include database connection data
var app = express();
var outputString = "";

app.engine('html', require('ejs').renderFile);

//[2] Get database connection data
app.locals.hostname = dbconfig.db['host'];
app.locals.username = dbconfig.db['username'];
app.locals.password = dbconfig.db['password'];
app.locals.port = dbconfig.db['port'];
app.locals.database = dbconfig.db['database'];
app.locals.connectionerror = 'successful';
app.locals.databases = '';

//[3] Connect to the Amazon RDS instance
var connection = mysql.createConnection({
    host: dbconfig.db['host'],
    user: dbconfig.db['username'],
    password: dbconfig.db['password'],
    port: dbconfig.db['port'],
    database: dbconfig.db['database']
});
```

```
connection.connect(function(err)
{
    if (err) {
        app.locals.connectionerror = err.stack;
        return;
    }
});

// [4] Query the database
connection.query('SHOW DATABASES', function (err, results) {
    if (err) {
        app.locals.databases = err.stack;
    }

    if (results) {
        for (var i in results) {
            outputString = outputString + results[i].Database + ', ';
        }
        app.locals.databases = outputString.slice(0, outputString.length-2);
    }
});

connection.end();

app.get('/', function(req, res) {
    res.render('./index.html');
});

app.use(express.static('public'));

//[5] Listen for incoming requests
app.listen(process.env.PORT);
```

The example displays the database connection information and also queries the database server and displays the server's databases. You can easily generalize it to interact with the database as needed. The following notes refer to the numbered comments in the preceding code.

[1] Include database connection data

This `require` statement includes the database connection data. As described later, when you attach a database instance to an app, OpsWorks Stacks puts the connection data in a file named `opsworks.js`, which looks similar to the following:

```
exports.db = {
  "host":"nodeexample.cdlqlk5uwd0k.us-west-2.rds.amazonaws.com",
  "database":"nodeexampledb",
  "port":3306,
  "username":"opsworksuser",
  "password":"your_pwd",
  "reconnect":true,
  "data_source_provider":"rds",
  "type":"mysql"}
```

opsworks.js is in the application's shared/config directory, /srv/www/*app_shortname*/shared/config. However, OpsWorks Stacks puts a symlink to opsworks.js in the application's root directory, so you can include the object by using just `require 'opsworks'`.

[2] Get database connection data

This set of statements displays the connection data from opsworks.js by assigning the values from the db object to a set of app.locals properties, each of which maps to one of the <%= ... %> elements in the index.html file. The rendered document replaces the <%= ... %> elements with the corresponding property values.

[3] Connect to the Amazon RDS instance

The example uses node-mysql to access the database. To connect to the database, the example creates a connection object by passing the connection data to createConnection, and then calls connection.connect to establish the connection.

[4] Query the database

After establishing a connection, the example calls connection.query to query the database. This example simply queries for the server's database names. query returns an array of results objects, one for each database, with the database name assigned to the Database property. The example concatenates the names and assigns them to app.locals.databases, which displays the list in the rendered HTML page.

For this example, there are five databases, the nodeexampledb database that you specified when you created the RDS instance and four others that are automatically created by Amazon RDS.

[5] Listen for incoming requests

The final statement listens for incoming requests on a specified port. You don't have to specify an explicit port value. When you add the app to your stack, you specify whether the application supports HTTP or HTTPS requests. OpsWorks Stacks then sets the `PORT` environment variable to 80 (HTTP) or 443 (HTTPS), and you can use that variable in your application.

It is possible to listen on other ports, but the Node.js App Server layer's built-in security group, **AWS-OpsWorks-nodejs-App-Server**, allows inbound user traffic only to ports 80, 443, and 22 (SSH). To allow inbound user traffic to other ports, create a security group with appropriate inbound rules and assign it to the Node.js App Server layer. Do not modify inbound rules by editing the built-in security group. Each time you create a stack, OpsWorks Stacks overwrites the built-in security groups with the standard settings, so any changes that you make will be lost.

> **Note**
>
> You can associate custom environment variables with your application when you create or update the associated app. You can also pass data to your application by using custom JSON and a custom recipe. For more information, see Passing Data to Applications.

Creating the Database Server and Load Balancer

This example uses Amazon RDS database server and Elastic Load Balancing load balancer instances. You must create each instance separately and then incorporate it into your stack. This section describes how to create new database and load balancer instances. You instead can use existing instances, but we recommend that you read through the procedure to ensure that those instances are correctly configured.

The following describes how to create a minimally configured RDS DB instance that is sufficient for this example. For more information, see the Amazon RDS User Guide.

To create the RDS DB instance

1. **Open the console.**

   Open the Amazon RDS console, and set the region to US West (Oregon). In the navigation pane, choose **RDS Dashboard**, and then choose **Launch DB Instance**.

2. **Specify the database engine.**

   Choose **MySQL Community Edition** as the database engine.

3. **Decline multi-AZ deployment.**

   Choose **No, this instance...**, and then choose **Next**. You don't need multi-AZ deployment for this example.

4. **Configure the basic settings.**

   On the **DB Instance Details** page, specify the following settings:

   - **DB Instance Class**: **db.t2.micro**
   - **Multi-AZ Deployment**: **No**
   - **Allocated Storage**: **5** GB
   - **DB Instance Identifier**: **`nodeexample`**
   - **Master Username**: **`opsworksuser`**
   - **Master Password**: A password of your choice

   Record the instance identifier, user name, and password for later use, accept the default settings for the other options, and then choose **Next**.

5. **Configure the advanced settings.**

   On the **Configure Advanced Settings** page, specify the following settings:

   - **Database Name**: **`nodeexampledb`**
   - **DB Security Group(s)**: **AWS-OpsWorks-DB-Master-Server**

   > **Note**
   >
   > The **AWS-OpsWorks-DB-Master-Server** security group allows only your stack's instances to access the database. If you want to access the database directly, attach an additional security group to the RDS DB instance with appropriate inbound rules. For more information, see Amazon RDS Security Groups. You also can control access by putting the instance in a VPC. For more information, see Running a Stack in a VPC.

Record the database name for later use, accept the default values for the other settings, and then choose **Launch DB Instance**.

The following procedure describes how to create an Elastic Load Balancing load balancer for this example. For more information, see the Elastic Load Balancing User Guide.

To create the load balancer

1. **Open the Amazon EC2 console.**

   Open the Amazon EC2 console and ensure that the region is set to US West (Oregon). In the navigation pane, choose **Load Balancers**, and then choose **Create Load Balancer**.

2. **Define the load balancer.**

   On the **Define Load Balancer** page, specify the following settings.

   - **Name** – **Node-LB**
   - **Create LB Inside** – **My Default VPC**

   Accept the default settings for the other options, and choose then **Next**.

3. **Assign security groups.**

   On the **Assign Security Groups** page, specify the following groups:

   - **default VPC security group**
   - **AWS-OpsWorks-nodejs-App-Server**

   Choose **Next**. On the **Configure Security Settings** page, choose **Next**. You don't need a secure listener for this example.

4. **Configure the health check.**

   On the **Configure Health Check** page, set **Ping Path** to **/** and accept the default values for the other settings. Choose **Next**. On the **Add EC2 Instances** page, choose **Next**. On the **Add Tags** page, choose **Review and Create**. OpsWorks Stacks handles the task of adding EC2 instances to the load balancer, and you won't need tags for this example.

5. **Create the load balancer.**

   On the **Review** page, choose **Create** to create the load balancer.

Creating the Stack

You now have all the components needed to create the stack.

To create the stack

1. **Sign in to the OpsWorks Stacks console.**

   Sign into the OpsWorks Stacks console, and choose **Add Stack**.

2. **Create the stack.**

   To create a new stack, choose **Chef 11 stack**, and then specify the following settings.

   - – **`NodeStack`**
   - **Region** – **US West (Oregon)**

     You can create a stack in any AWS region, but we recommend US West (Oregon) for tutorials.

   Choose **Add Stack**. For more information on stack configuration settings, see Create a New Stack.

3. **Add a Node.js App Server layer with an attached load balancer.**

   On the **NodeStack** page, choose **Add a layer**, and then specify the following settings:

   - **Layer type** – **Node.js App Server**
   - **Elastic Load Balancer** – **Node-LB**

   Accept the default values for the other settings, and then choose **Add Layer**.

4. **Add instances to the layer and start them.**

   In the navigation pane, choose **Instances**, and then add two instances to the Rails App Server layer, as follows.

   1. Under **Node.js App Server**, choose **Add instance**.

Set **Size** to **t2.micro**, accept the default values for the other settings, and then choose **Add Instance**.

2. Choose **+Instance**, and then add a second **t2.micro** instance to the layer in a different subnet.

   This places the instance in a different Availability Zone (AZ).

3. Choose **Add instance**.
4. To start both instances, choose **Start All Instances**.

You have assigned an Elastic Load Balancing load balancer to this layer. When an instance enters or leaves the online state, OpsWorks Stacks automatically registers or deregisters the instance with the load balancer.

> **Note**
>
> For a production stack, we recommend that you distribute your application server instances across multiple AZs. If users can't connect to an AZ, the load balancer routes incoming traffic to instances in the remaining zones, and your site will continue to function.

5. **Register the RDS DB instance with the stack.**

   In the navigation pane, choose **Resources** and register the RDS DB instance with the stack, as follows.

   1. Choose the **RDS** tab, and then choose **Show Unregistered RDS DB** instances.
   2. Choose the **nodeexampledb** instance, and then specify the following settings:
      - **User** – The master user name that you specified when you created the instance; for this example. **`opsworksuser`**.
      - **Password** – The master password that you specified when you created the instance.
   3. Choose **Register with Stack** to add the RDS DB instance to the stack as an Amazon RDS service layer.

> **Warning**
>
> OpsWorks Stacks does not validate the **User** or **Password** values, it simply passes them to the application. If you enter them incorrectly, your application cannot connect to the database.

To add the RDS DB instance to the stack as an Amazon RDS service layer, choose **Register with Stack**.

Deploying the Application

You must store the application in a remote repository. When you deploy it, OpsWorks Stacks deploys the code and related files from the repository to the application server instances. For convenience, this example uses a public Amazon Simple Storage Service (Amazon S3) archive as the repository, but you also can use several other repository types, including Git and Subversion. For more information, see Application Source.

To deploy the application

1. **Package the application in an archive file.**

   Create a `.zip` archive of the nodedb directory and subdirectories named nodedb.zip. You also can use other types of archive file, including gzip, bzip2, and tarball. Note that OpsWorks Stacks does not support uncompressed tarballs. For more information, see Application Source.

2. **Upload the archive file to Amazon S3.**

   Upload `nodedb.zip` to an Amazon S3 bucket, make the file public, and copy the file's URL for later use. For more information on how to create buckets and upload files, go to Get Started With Amazon Simple Storage Service.

   > **Note**
   >
   > OpsWorks Stacks can also deploy private files from an Amazon S3 bucket, but for simplicity, this example uses a public file. For more information, see Application Source.

3. **Create an OpsWorks Stacks app.**

   Return to the OpsWorks Stacks console, in the navigation pane, choose **Apps**, and then choose **Add an app**. Specify the following settings:

   - **Name** – `NodeDB`.

     This string is the app's display name. For most purposes, you need the app's short name, which OpsWorks Stacks generates from the display name by transforming all characters to lower case and removing punctuation. For this example, the short name is `nodedb`. To verify an app's short name, after creating the app, choose the app on the **Apps** page to display its details page.
   - **Type** – `Node.js`.
   - **Data source type** – `RDS`.
   - **Database instance** – Choose the Amazon RDS DB instance that you registered earlier.
   - **Database name** – Specify the database name that you created earlier, `nodeexampledb` for this example.
   - **Repository type** – `Http Archive`.

     You must use this repository type for public Amazon S3 files. The `S3 Archive` type is used only for private archives.
   - **Repository URL** – The archive file's Amazon S3 URL.

   Use the default values for the remaining settings, and then click **Add App** to create the app.

4. **Deploy the app.**

   Go to the **Apps** page, and in the NodeDB app's **Actions** column, choose **deploy**. Then choose **Deploy** to deploy the app to the server instances. OpsWorks Stacks runs the Deploy recipes on each instance, which downloads the application from the repository and restarts the server. When each instance has a green check mark and the **Status** is **successful**, deployment is complete and the application is ready to start handling requests.

   > **Note**
   >
   > If the deployment fails, choose **show** in the **Log** column to display the deployment's Chef log. The error information is near the bottom.

5. **Open the application.**

   To open the application, choose **Layers**, choose the load balancer, and then choose the load balancer's DNS name, which sends an HTTP request to the load balancer. You should see something like the following.

   > # AWS OpsWorks Node.js Example
   >
   > Amazon RDS Endpoint: *nodeexample.cdlqlk5uwd0k.us-west-2.rds.amazonaws.com*
   >
   > User: *opsworksuser*
   >
   > Password: *Your-Pwd*
   >
   > Port: *3306*
   >
   > Database: *nodeexampledb*
   >
   > Connection: successful
   >
   > Databases: information_schema, innodb, mysql, nodeexampledb, performance_schema

Note

OpsWorks Stacks automatically deploys apps to new instances during setup. Manual deployment is required only for online instances. For more information, see Deploying Apps. For a general discussion of deployment, including some more sophisticated deployment strategies, see Managing and Deploying Apps and Cookbooks.

What Next?

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This walkthrough took you through the basics of setting up a simple Node.js application server stack. Here are some suggestions for what to do next.

Examine the Node.js built-in cookbooks

If you want to know how the instances are configured in detail, see the layer's built-in cookbook, opsworks_nodejs, which contains the recipes and related files that OpsWorks Stacks uses to install and configure the software, and the built-in deploy cookbook, which contains the recipes that OpsWorks Stacks uses to deploy the apps.

Customize the server configuration

The example stack is fairly basic. For production use, you will probably want to customize the stack. For more information, see Customizing OpsWorks Stacks.

Add SSL support

You can enable SSL support for your app and provide OpsWorks Stacks with the appropriate certificates when you create the app. OpsWorks Stacks then installs the certificates in the appropriate directory. For more information, see Using SSL.

Add in-memory data caching

Production-level sites often improve performance by caching data in an in-memory key-value store, such as Redis or Memcache. You can use either with an OpsWorks Stacks stack. For more information, see ElastiCache Redis and Memcached.

Use a more sophisticated deployment strategy

The example used a simple app deployment strategy, which deploys the update to every instance concurrently. This approach is simple and fast, but there is no margin for error. If the deployment fails or the update has any issues, every instance in your production stack could be affected, potentially disrupting or disabling your site until you can fix the problem. For more information on deployment strategies, see Managing and Deploying Apps and Cookbooks.

Extend the Node.js App Server layer

You can extend the layer in a variety of ways. For example, you can implement recipes to run scripts on the instances or implement Chef deployment hooks to customize app deployment. For more information, see Extending a Layer.

Define environment variables

You can pass data to your application by defining environment variables for the associated app. When you deploy the app, OpsWorks Stacks exports those variables so you can access them from your app. For more information, see Using Environment Variables.

Customizing OpsWorks Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks built-in layers provide standard functionality that is sufficient for many purposes. However, you might encounter one or more of the following:

- A built-in layer's standard configuration is adequate but not ideal; you would like to optimize it for your particular requirements.

  For example, you might want to tune a Static Web Server layer's Nginx server configuration by specifying your own values for settings such as the maximum number of worker processes or the `keepalivetimeout` value.
- A built-in layer's functionality is fine, but you want to extend it by installing additional packages or running some custom installation scripts.

  For example, you might want to extend a PHP App Server layer by also installing a Redis server.
- You have requirements that aren't handled by any of the built-in layers.

  For example, OpsWorks Stacks does not include built-in layers for some popular database servers. You can create a custom layer that installs those servers on the layer's instances.
- You are running a Windows stack, which support only custom layers.

OpsWorks Stacks provides a variety of ways to customize layers to meet your specific requirements. The following examples are listed in order of increasing complexity and power:

> **Note**
>
> Some of these approaches work only for Linux stacks. See the following topics for details.

- Use custom JSON to override default OpsWorks Stacks settings.
- Implement a custom Chef cookbook with an attributes file that overrides the default OpsWorks Stacks settings.
- Implement a custom Chef cookbook with a template that overrides or extends a default OpsWorks Stacks template.
- Implement a custom Chef cookbook with a simple recipe that runs a shell script.
- Implement a custom Chef cookbook with recipes that perform tasks such as creating and configuring directories, installing packages, creating configuration files, deploying apps, and so on.

You can also override recipes, depending on the stack's Chef version and operating system.

- With Chef 0.9 and 11.4 stacks, you cannot override a built-in recipe by implementing a custom recipe with the same cookbook and recipe name.

  For each lifecycle event, OpsWorks Stacks always runs the built-in recipes first, followed by any custom recipes. Because these Chef versions do not run a recipe with the same cookbook and recipe name twice, the built-in recipe takes precedence and the custom recipe is not executed.
- You can override built-in recipes on Chef 11.10 stacks.

  For more information, see Cookbook Installation and Precedence.
- You cannot override built-in recipes on Windows stacks.

  The way that OpsWorks Stacks handles Chef runs for Windows stacks does not allow built-in recipes to be overridden.

> **Note**
>
> Because many of the techniques use custom cookbooks, you should first read Cookbooks and Recipes if you are not already familiar with cookbook implementation. Cookbook Basics provides a detailed tutorial introduction to implementing custom cookbooks, and Implementing Cookbooks for OpsWorks Stacks covers some of the details about how to implement cookbooks for OpsWorks Stacks instances.

Topics

- Customizing OpsWorks Stacks Configuration by Overriding Attributes
- Extending OpsWorks Stacks Configuration Files Using Custom Templates
- Extending a Layer
- Creating a Custom Tomcat Server Layer
- Stack Configuration and Deployment Attributes

Customizing OpsWorks Stacks Configuration by Overriding Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> For Windows stacks and Chef 12 Linux stacks, OpsWorks Stacks uses separate Chef runs for built-in recipes and custom recipes. This means that you cannot use the techniques discussed in this section to override built-in attributes for Windows stacks and Chef 12 Linux stacks.

Recipes and templates depend on a variety of Chef attributes for instance or stack-specific information such as layer configurations or application server settings. These attributes have several sources:

- **Custom JSON**–You can optionally specify custom JSON attributes when you create, update, or clone a stack, or when you deploy an app.
- **Stack configuration attributes**–OpsWorks Stacks defines these attributes to hold stack configuration information, including the information that you specify through the console settings.
- **Deployment attributes**–AWS OpsWorks defines deployment-related attributes for Deploy events.
- **Cookbook attributes**– Built-in and custom Cookbooks usually include one or more attribute files, which contain attributes that represent cookbook-specific values such as application server configuration settings.
- **Chef**–Chef's Ohai tool defines attributes that represent a wide variety of system configuration settings, such as CPU type and installed memory.

For a complete list of stack configuration and deployment attributes and built-in cookbook attributes , see Stack Configuration and Deployment Attributes: Linux and Built-in Cookbook Attributes. For more information about Ohai attributes, see Ohai.

When a lifecycle event such as Deploy or Configure occurs, or you run a stack command such as `execute_recipes` or `update_packages`, OpsWorks Stacks does the following:

- Sends a corresponding command to the agent on each affected instance.

  The agent runs the appropriate recipes. For example, for a Deploy event, the agent runs the built-in Deploy recipes, followed by any custom Deploy recipes.
- Merges any custom JSON and deployment attributes with the stack configuration attributes and installs them on the instances.

The attributes from custom JSON, stack configuration and deployment attributes, cookbook attributes, and Ohai attributes are merged into a *node object*, which supplies attribute values to recipes. An instance is essentially stateless as far as stack configuration attributes are concerned, including any custom JSON. When you run a deployment or stack command, the associated recipes use the stack configuration attributes that were downloaded with the command.

Topics

- Attribute Precedence
- Overriding Attributes With Custom JSON
- Overriding OpsWorks Stacks Attributes Using Custom Cookbook Attributes

Attribute Precedence

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

If an attribute is uniquely defined, Chef simply incorporates it into the node object. However, any attribute source can define any attribute, so it is possible for the same attribute to have multiple definitions with different values. For example, the built-in `apache2` cookbook defines `node[:apache][:keepalive]`, but you could also define that attribute in custom JSON or in a custom cookbook. If an attribute has multiple definitions, they are evaluated in an order that is described later and the node object receives the definition with the highest precedence.

An attribute is defined as follows:

```
node.type[:attribute][:sub_attribute][:...]=value
```

If an attribute has multiple definitions, the type determines which definition has precedence, and that definition is incorporated into the node object. OpsWorks Stacks uses the following attribute types:

- **default**–This is the most common type, and it essentially means "use this value if the attribute hasn't already been defined." If all definitions of an attribute are `default` type, the first definition in the evaluation order has precedence and subsequent values are ignored. Note that OpsWorks Stacks sets all stack configuration and deployment attribute definitions to `default` type.

- **normal**–Attributes with this type override any `default` or `normal` attributes that were defined earlier in the evaluation order. For example, if the first attribute is from a built-in cookbook and has a `default` type and the second is a user-defined attribute with has a `normal` type, the second definition has precedence.
- **set**–This is a deprecated type that you might see in older cookbooks. It has been superseded by `normal`, which has the same precedence.

Chef supports several additional attribute types, including an `automatic` type that takes precedence over all other attribute definitions. The attribute definitions generated by Chef's Ohai tool are all `automatic` types, so they are effectively read-only. This isn't usually an issue, because there is no reason to override them and they are distinct from OpsWorks Stacks' attributes. However, you should be careful to name your custom cookbook attributes so they are distinct from the Ohai attributes. For more information, see About Attributes.

> **Note**
>
> The Ohai tool is an executable that you can run from the command line. To list an instance's Ohai attributes, log in to the instance and run `ohai` in a terminal window. Be aware that it produces a very long output.

Here are the steps that incorporate the various attribute definitions into the node object:

1. Merge any custom stack configuration attributes into the stack configuration and deployment attributes.

   Custom JSON attributes can be set for the stack, or for a particular deployment. They are first in the evaluation order and are effectively `normal` types. If one or more stack configuration attributes are also defined in custom JSON, the custom JSON values take precedence. Otherwise OpsWorks Stacks simply incorporates the custom JSON attributes into the stack configuration.
2. Merge any deployment custom JSON attributes into the stack configuration and deployment attributes.

   Deployment custom JSON attributes are also effectively `normal` types, so they take precedence over built-in and custom stack configuration JSON and built-in deployment JSON.
3. Merge the stack configuration and deployment attributes into the instance's node object.
4. Merge the instance's built-in cookbook attributes into the node object.

The built-in cookbook attributes are all `default` types. If the one or more built-in cookbook attributes are also defined in the stack configuration and deployment attributes—typically because you defined them with custom JSON—the stack configuration definitions take precedence over the built-in cookbook definitions. All other built-in cookbook attributes are simply incorporated into the node object.

5. Merge the instance's custom cookbook attributes into the node object.

   Custom c ookbook attributes are usually either `normal` or `default` types. Unique attributes are incorporated into the node object. If any custom cookbook attributes are also defined in Steps 1–3 (typically because you defined them with custom JSON), precedence depends on the custom cookbook attribute's type:

   - Attributes defined in Steps 1–3 take precedence over custom cookbook `default` attributes.
   - Custom c ookbook `normal` attributes take precedence over definitions from Steps 1–3.

> **Important**
>
> Do not use custom cookbook `default` attributes to override stack configuration or built-in cookbook attributes. Because custom cookbook attributes are evaluated last, the `default` attributes have the lowest precedence, and cannot override anything.

Overriding Attributes With Custom JSON

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> Because OpsWorks Stacks handles Chef runs differently for Windows stacks than for Linux stacks, you cannot use the techniques discussed in this section for Windows stacks.

The simplest way to override an OpsWorks Stacks attribute is to define it in custom JSON, which takes precedence over stack configuration and deployment attributes as well as built-in and custom cookbook `default` attributes. For more information, see Attribute Precedence.

> **Important**
>
> You should override stack configuration and deployment attributes with care. For example overriding attributes in the `opsworks` namespace can interfere with the built-in recipes. For more information, see Stack Configuration and Deployment Attributes.

You can also use custom JSON to define unique attributes, typically to pass data to your custom recipes. The attributes are simply incorporated into the node object, and recipes can reference them by using the standard Chef node syntax.

How to Specify Custom JSON

To use custom JSON to override an attribute value, you must first determine the attribute's fully qualified attribute name. You then create a JSON object that contains the attributes you want to override, set to your preferred values. For convenience, Stack Configuration and Deployment Attributes: Linux and Built-in Cookbook Attributes documents commonly used stack configuration, deployment, and built-in cookbook attributes, including their fully qualified names.

The object's parent-child relationships must correspond to the appropriate fully qualified Chef nodes. For example, suppose you want to change the following Apache attributes:

- The `keepalivetimeout` attribute, whose node is `node[:apache][:keepalivetimeout]` and has a default value of `3`.
- The `logrotate` `schedule` attribute, whose node is `node[:apache][:logrotate][:schedule]`, and has a default value of `"daily"`.

To override the attributes and set the values to 5 and "weekly", respectively, you would use the following custom JSON:

```
{
  "apache" : {
    "keepalivetimeout" : 5,
    "logrotate" : {
       "schedule" : "weekly"
    }
  }
}
```

When to Specify Custom JSON

You can specify a custom JSON structure for the following tasks:

- Create a new stack
- Update a stack
- Run a stack command
- Clone a stack
- Deploy an app

For each task, OpsWorks Stacks merges the custom JSON attributes with the stack configuration and deployment attributes and sends it to the instances, to be merged into the node object. However, note the following:

- If you specify custom JSON when you create, clone, or update a stack, the attributes are merged into the stack configuration and deployment attributes for all subsequent lifecycle events and stack commands.
- If you specify custom JSON for a deployment, the attributes are merged into the stack configuration and deployment attributes only for the corresponding event.

  If you want to use those custom attributes for subsequent deployments, you must explicitly specify the custom JSON again.

It is important to remember that attributes only affect the instance when they are used by recipes. If you override an attribute value but no subsequent recipes reference the attribute, the change has

no effect. You must either ensure that the custom JSON is sent before the associated recipes run, or ensure that the appropriate recipes are re-run.

Custom JSON Best Practices

You can use custom JSON to override any OpsWorks Stacks attribute, but manually entering the information is somewhat cumbersome, and it is not under any sort of source control. Custom JSON is best used for the following purposes:

- When you want to override only a small number of attributes, and you do not otherwise need to use custom cookbooks.

  With custom JSON, you can avoid the overhead of setting up and maintaining a cookbook repository just to override a couple of attributes.
- Sensitive values, such as passwords or authentication keys.

  Cookbook attributes are stored in a repository, so any sensitive information is at some risk of being compromised. Instead, define attributes with dummy values and use custom JSON to set the real values.
- Values that are expected to vary.

  For example, a recommended practice is to have your production stack supported by separate development and staging stacks. Suppose that these stacks support an application that accepts payments. If you use custom JSON to specify the payment endpoint, you can specify a test URL for your staging stack. When you are ready to migrate an updated stack to your production stack, you can use the same cookbooks and use custom JSON to set the payment endpoint to the production URL.
- Values that are specific to a particular stack or deployment command.

Overriding OpsWorks Stacks Attributes Using Custom Cookbook Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> For Windows stacks, OpsWorks Stacks uses separate Chef runs for built-in recipes and custom recipes. This means that you cannot use the techniques discussed in this section to override built-in attributes for Windows stacks.

Custom JSON is a convenient way to override OpsWorks Stacks stack configuration and built-in cookbook attributes, but it has some limitations. In particular, you must enter custom JSON manually for each use, so you have no robust way to manage the definitions. A better approach is often to use custom cookbook attribute files to override built-in attributes. Doing so allows you to place the definitions under source control.

The procedure for using custom attribute files to override OpsWorks Stacks definitions is straightforward.

To override OpsWorks Stacks attribute definitions

1. Set up a cookbook repository, as described in Cookbooks and Recipes.
2. Create a cookbook with the same name as the built-in cookbook that contains the attributes that you want to override. For example, to override the Apache attributes, the cookbook should be named apache2.
3. Add an `attributes` folder to the cookbook and add a file to that folder named `customize.rb`.
4. Add an attribute definition to the file for each of the built-in cookbook's attributes that you want to override, set to your preferred value. The attribute must be a `normal` type or higher and have exactly the same node name as the corresponding OpsWorks Stacks attribute. For a detailed list of OpsWorks Stacks attributes, including node names, see Stack Configuration and Deployment Attributes: Linux and Built-in Cookbook Attributes. For more information on attributes and attributes files, see About Attribute Files.

   > **Important**
   >
   > Your attributes must be `normal` type to override OpsWorks Stacks attributes; `default` types do not have precedence. For example, if your `customize.rb` file contains a `default[:apache][:keepalivetimeout] = 5` attribute definition, the

corresponding attribute in the built-in `apache.rb` attributes file is evaluated first, and takes precedence. For more information, see Overriding Attributes.

5. Repeat Steps 2 – 4 for each built-in cookbook with attributes that you want to override.
6. Enable custom cookbooks for your stack and provide the information required for OpsWorks Stacks to download your cookbooks to the stack's instances. For more information, see Installing Custom Cookbooks.

Note

For a complete walkthrough of this procedure, see Overriding Built-In Attributes.

The node object used by subsequent lifecycle events, deploy commands, and stack commands will now contain your attribute definitions instead of the OpsWorks Stacks values.

For example, to override the built-in Apache `keepalivetimeout` and `logrotate schedule` settings discussed in How to Specify Custom JSON, add an `apache2` cookbook to your repository and add a `customize.rb` file to the cookbook's `attributes` folder with the following contents.

```
normal[:apache][:keepalivetimeout] = 5
normal[:apache][:logrotate][:schedule] = 'weekly'
```

Important

You should not override OpsWorks Stacks attributes by modifying a copy of the associated built-in attributes file. If, for example, you copy `apache.rb` to your `apache2/attributes` folder and modify some of its settings, you essentially override every attribute in the built-in file. Recipes will use the attribute definitions from your copy and ignore the built-in file. If OpsWorks Stacks later modifies the built-in attributes file, recipes will not have access to the changes unless you manually update your copy.
To avoid this situation, all built-in cookbooks contain an empty `customize.rb` attributes file, which is required in all modules through an `include_attribute` directive. By overriding attributes in your copy of `customize.rb`, you affect only those specific attributes. Recipes will obtain any other attribute values from the built-in attributes files, and automatically get the current values of any attributes that you have not overridden.

This approach helps you to keep the number of attributes in your cookbook repository small, which reduces your maintenance overhead and makes future upgrades easier to manage.

Extending OpsWorks Stacks Configuration Files Using Custom Templates

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

Because OpsWorks Stacks handles Chef runs differently for Windows stacks than for Linux stacks, you cannot use the techniques discussed in this section for Windows stacks.

OpsWorks Stacks uses templates to create files such as configuration files, which typically depend on attributes for many of the settings. If you use custom JSON or custom cookbook attributes to override the OpsWorks Stacks definitions, your preferred settings are incorporated into the configuration files in place of the OpsWorks Stacks settings. However, OpsWorks Stacks does not necessarily specify an attribute for every possible configuration setting; it accepts the defaults for some settings and hardcodes others directly in the template. You can't use custom JSON or custom cookbook attributes to specify preferred settings if there is no corresponding OpsWorks Stacks attribute.

You can extend the configuration file to include additional configuration settings by creating a custom template. You can then add whatever configuration settings or other content you need to the file, and override any hardcoded settings. For more information on templates, see Templates.

Note

You can override any built-in template *except* opsworks-agent.monitrc.erb.

To create a custom template

1. Create a cookbook with the same structure and directory names as the built-in cookbook. Then, create a template file in the appropriate directory with the same name as the built-in template that you want to customize. For example, to use a custom template to extend the Apache `httpd.conf` configuration file, you must implement an `apache2` cookbook in your repository and your template file must be `apache2/templates/default/apache.conf.erb`. Using exactly the same names allows OpsWorks Stacks to recognize the custom template and use it instead of the built-in template.

   The simplest approach is to just copy the built-in template file from the built-in cookbook's GitHub repository to your cookbook and modify it as needed.

   > **Important**
   >
   > Do not copy any files from the built-in cookbook except for the template files that you want to customize. Copies of other types of cookbook file, such as recipes, create duplicate Chef resources and can cause errors.

   The cookbook can also include custom attributes, recipes, and related files, but their file names should not duplicate built-in file names.
2. Customize the template file to produce a configuration file that meets your requirements. You can add more settings, delete existing settings, replace hardcoded attributes, and so on.
3. If you haven't done so already, edit the stack settings to enable custom cookbooks and specify your cookbook repository. For more information, see Installing Custom Cookbooks.

> **Note**
>
> For a complete walkthrough of this procedure, see Overriding Built-In Templates.

You don't have to implement any recipes or add recipes to the layer configuration to override a template. OpsWorks Stacks always runs the built-in recipes. When it runs the recipe that creates the configuration file, it will automatically use your custom template instead of the built-in template.

Note

If OpsWorks Stacks makes any changes to the built-in template, your custom template might become out of sync and no longer work correctly. For example, suppose your template refers to a dependent file, and the file name changes. OpsWorks Stacks doesn't make such changes often, and when a template does change, it lists the changes and gives you the option of upgrading to a new version. You should monitor the OpsWorks Stacks repository for changes, and manually update your template as needed.

Extending a Layer

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Sometimes, you need to customize a built-in layer beyond what can be handled by modifying OpsWorks Stacks attributes or customizing templates. For example, suppose you need to create symlinks, set file or folder modes, install additional packages, and so on. You must extend custom layers to provide more than minimal functionality. In that case, you will need to implement one or more custom cookbooks with recipes to handle the customization tasks. This topic provides some examples of how to use recipes to extend a layer.

If you are new to Chef, you should first read Cookbooks 101, which is a tutorial that introduces the basics of how to implement cookbooks to perform a variety of common tasks. For a detailed example of how to implement a custom layer, see Creating a Custom Tomcat Server Layer.

Topics

- Using Recipes to Run Scripts
- Using Chef Deployment Hooks
- Running Cron Jobs on Linux Instances
- Installing and Configuring Packages on Linux Instances

Using Recipes to Run Scripts

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

If you already have a script that performs the required customization tasks, the simplest approach to extending a layer is often to implement a simple recipe to run the script. You can then assign the recipe to the appropriate lifecycle events, typically Setup or Deploy, or use the `execute_recipes` stack command to run the recipe manually.

The following example runs a shell script on Linux instances, but you can use the same approach for other types of script, including Windows PowerShell scripts.

```
cookbook_file "/tmp/lib-installer.sh" do
  source "lib-installer.sh"
  mode 0755
end

execute "install my lib" do
  command "sh /tmp/lib-installer.sh"
end
```

The `cookbook_file` resource represents a file that is stored in a subdirectory of a cookbook's `files` directory, and transfers the file to a specified location on the instance. This example transfers a shell script, `lib-installer.sh`, to the instance's `/tmp` directory and sets the file's mode to `0755`. For more information, see cookbook_file.

The `execute` resource represents a command, such as a shell command. This example runs `lib-installer.sh`. For more information, see execute.

You can also run a script by incorporating it into a recipe. The following example runs a bash script, but Chef also supports Csh, Perl, Python, and Ruby.

```
script "install_something" do
  interpreter "bash"
  user "root"
  cwd "/tmp"
  code <<-EOH
    #insert bash script
  EOH
end
```

The `script` resource represents a script. The example specifies a bash interpreter, sets user to `"root"`, and sets the working directory to `/tmp`. It then runs the bash script in the `code` block, which can include as many lines as required. For more information, see script.

For more information on how to use recipes to run scripts, see Example 7: Running Commands and Scripts. For an example of how to run a PowerShell script on a Windows instance, see Running a Windows PowerShell Script.

Using Chef Deployment Hooks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can customize deployment by implementing a custom recipe to perform the required tasks and assigning it to the appropriate layer's Deploy event. An alternative and sometimes simpler approach—especially if you don't need to implement a cookbook for other purposes—is to use Chef deployment hooks to run your customization code. In addition, custom Deploy recipes run after the deployment has already been performed by the built-in recipes. Deployment hooks allow you to interact during a deployment, for example, after the app's code is checked out of the repository but before Apache is restarted.

Chef deploys apps in four stages:

- **Checkout**–Downloads the files from the repository

- **Migrate**–Runs a migration, as required
- **Symlink**–Creates symlinks
- **Restart**–Restarts the application

Chef deployment hooks provide a simple way to customize a deployment by optionally running a user-supplied Ruby application after each stage completes. To use deployment hooks, implement one or more Ruby applications and place them in your app's `/deploy` directory. (If your app does not have a `/deploy` directory, create one at the `APP_ROOT` level.) The application must have one of the following names, which determines when it runs.

- `before_migrate.rb` runs after the Checkout stage is complete but before Migrate.
- `before_symlink.rb` runs after the Migrate stage is complete but before Symlink.
- `before_restart.rb` runs after the Symlink stage is complete but before Restart.
- `after_restart.rb` runs after the Restart stage is complete.

Chef deployment hooks can access the node object by using standard node syntax, just like recipes. Deployment hooks can also access the values of any app environment variables that you have specified. However, you must use `new_resource.environment["`*`VARIABLE_NAME`*`"]` to access the variable's value instead of `ENV["`*`VARIABLE_NAME`*`"]`.

Running Cron Jobs on Linux Instances

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

A Linux cron job directs the cron daemon to run one or more commands on a specified schedule. For example, suppose your stack supports a PHP e-commerce application. You can set up a cron job to have the server send you a sales report at a specified time every week. For more information about cron, see cron on Wikipedia. For more information about how to run a cron job directly on a

Linux-based computer or instance, see What are cron and crontab, and how do I use them? on the Indiana University knowledge base website.

Although you can manually set up `cron` jobs on individual Linux-based instances by connecting to them with SSH, and editing their `crontab` entries, a key advantage of OpsWorks Stacks is that you can direct it to run the task across an entire layer of instances. The following procedure describes how to set up a `cron` job on a PHP App Server layer's instances, but you can use the same approach with any layer.

To set up a `cron` job on a layer's instances

1. Implement a cookbook with a recipe with a `cron` resource that sets up the job. The example assumes that the recipe is named `cronjob.rb`; the implementation details are described later. For more information on cookbooks and recipes, see Cookbooks and Recipes.
2. Install the cookbook on your stack. For more information, see Installing Custom Cookbooks.
3. Have OpsWorks Stacks run the recipe automatically on the layer's instances by assigning it to the following lifecycle events. For more information, see Automatically Running Recipes.

   - **Setup** – Assigning `cronjob.rb` to this event directs OpsWorks Stacks to run the recipe on all new instances.
   - **Deploy** – Assigning `cronjob.rb` to this event directs OpsWorks Stacks to run the recipe on all online instances when you deploy or redeploy an app to the layer.

   You can also manually run the recipe on online instances by using the `Execute Recipes` stack command. For more information, see Run Stack Commands.

The following is the `cronjob.rb` example, which sets up a cron job to run a user-implemented PHP application once a week that collects the sales data from the server and mails a report. For more examples of how to use a cron resource, see cron.

```
cron "job_name" do
  hour "1"
  minute "10"
  weekday "6"
  command "cd /srv/www/myapp/current && php .lib/mailing.php"
end
```

`cron` is a Chef resource that represents a `cron` job. When OpsWorks Stacks runs the recipe on an instance, the associated provider handles the details of setting up the job.

- *`job_name`* is a user-defined name for the `cron` job, such as `weekly report`.
- `hour`/`minute`/`weekday` specify when the commands should run. This example runs the commands every Saturday at 1:10 AM.
- `command` specifies the commands to be run.

  This example runs two commands. The first navigates to the `/srv/www/myapp/current` directory. The second runs the user-implemented `mailing.php` application, which collects the sales data and sends the report.

Note

The `bundle` command does not work with `cron` jobs by default. The reason is that OpsWorks Stacks installs bundler in the `/usr/local/bin` directory. To use `bundle` with a `cron` job, you must explicitly add the path `/usr/local/bin` to the cron job. Also, because the $PATH environment variable may not expand in the `cron` job, a best practice is to explicitly add any necessary path information to the job without relying on expansion of the $PATH variable. The following examples show two ways to use `bundle` in a `cron` job.

```
cron "my first task" do
  path "/usr/local/bin"
  minute "*/10"
  command "cd /srv/www/myapp/current && bundle exec my_command"
end
```

```
cron_env = {"PATH" => "/usr/local/bin"}
cron "my second task" do
  environment cron_env
  minute "*/10"
  command "cd /srv/www/myapp/current && /usr/local/bin/bundle exec my_command"
end
```

If your stack has multiple application servers, assigning `cronjob.rb` to the PHP App Server layer's lifecycle events might not be an ideal approach. For example, the recipe runs on all of the layer's instances, so you will receive multiple reports. A better approach is to use a custom layer to ensure that only one server sends a report.

To run a recipe on just one of a layer's instances

1. Create a custom layer called, for example, PHPAdmin and assign `cronjob.rb` to its Setup and Deploy events. Custom layers don't necessarily have to do very much. In this case, PHPAdmin just runs one custom recipe on its instances.
2. Assign one of the PHP App Server instances to AdminLayer. If an instance belongs to more than one layer, OpsWorks Stacks runs each layer's built-in and custom recipes.

Because only one instance belongs to the PHP App Server and PHPAdmin layers, `cronjob.rb` runs only on that instance and you receive just one report.

Installing and Configuring Packages on Linux Instances

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The built-in layers support only certain packages. For more information, see Layers. You can install other packages, such as a Redis server, by implementing custom recipes to handle the associated setup, configuration, and deployment tasks. In some cases, the best approach is to extend a built-in layer to have it install the package on its instances alongside the layer's standard packages. For example, if you have a stack that supports a PHP application, and you would like to include a Redis server, you could extend the PHP App Server layer to install and configure a Redis server on the layer's instances in addition to a PHP application server.

A package installation recipe typically needs to perform tasks like these:

- Create one or more directories and set their modes.

- Create a configuration file from a template.
- Run the installer to install the package on the instance.
- Start one or more services.

For an example of how to install a Tomcat server, see Creating a Custom Tomcat Server Layer. The topic describes how to set up a custom Redis layer, but you could use much the same code to install and configure Redis on a built-in layer. For examples of how to install other packages, see the built-in cookbooks, at https://github.com/aws/opsworks-cookbooks.

Creating a Custom Tomcat Server Layer

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This topic describes how to implement a custom layer for a Linux stack. However, the basic principles and some of the code can also be adapted to implement custom layers for Windows stacks, especially those in the section on app deployment.

The simplest way to use nonstandard packages on OpsWorks Stacks instances is to extend an existing layer. However, this approach installs and runs both the standard and nonstandard packages on the layer's instances, which is not always desirable. A somewhat more demanding but more powerful approach is to implement a custom layer, which gives you almost complete control over the layer's instances, including the following:

- Which packages are installed
- How each package is configured
- How to deploy apps from a repository to the instance

Whether using the console or API, you create and manage a custom layer much like any other layer, as described in Custom Layers. However, a custom layer's built-in recipes perform only some very basic tasks, such as installing a Ganglia client to report metrics to a Ganglia master. To make a custom layer's instances more than minimally functional, you must implement one or more custom cookbooks with Chef recipes and related files to handle the tasks of installing and configuring packages, deploying apps, and so on. You don't necessarily have to implement everything from scratch, though. For example, if you store applications in one of the standard repositories, you can use the built-in deploy recipes to handle much of the work of installing the applications on the layer's instances.

Note

If you are new to Chef, you should first read Cookbooks 101, which is a tutorial that introduces the basics of how to implement cookbooks to perform a variety of common tasks.

The following walkthrough describes how to implement a custom layer that supports a Tomcat application server. The layer is based on a custom cookbook named Tomcat, which includes recipes to handle package installation, deployment, and so on. The walkthrough includes excerpts from the Tomcat cookbook. You can download the complete cookbook from its GitHub repository. If you are not familiar with Opscode Chef, you should first read Cookbooks and Recipes.

Note

OpsWorks Stacks includes a full-featured Java App Server layer for production use. The purpose of the Tomcat cookbook is to show how to implement custom layers, so it supports only a limited version of Tomcat that does not include features such as SSL. For an example of a full featured implementation, see the built-in opsworks_java cookbook .

The Tomcat cookbook supports a custom layer whose instances have the following characteristics:

- They support a Tomcat Java application server with an Apache front end.
- Tomcat is configured to allow applications to use a JDBC `DataSource` object to connect to a separate MySQL instance, which serves as a back end data store.

The cookbook for this project involves several key components:

- Attributes file contains configuration settings that are used by the various recipes.
- Setup recipes are assigned to the layer's Setup lifecycle event. They run after an instance has booted and perform tasks such as installing packages and creating configuration files.
- Configure recipes are assigned to the layer's Configure lifecycle event. They run after the stack's configuration changes—primarily when instances come online or go offline—and handle any required configuration changes.
- Deploy recipes are assigned to the layer's Deploy lifecycle event. They run after the Setup recipes and when you manually deploy an app to install the code and related files on a layer's instances and handle related tasks, such as restarting services.

The final section, Create a Stack and Run an Application, describes how to create a stack that includes a custom layer based on the Tomcat cookbook and how to deploy and run a simple JSP application that displays data from a MySQL database running on an instance that belongs to a separate MySQL layer.

> **Note**
>
> The Tomcat cookbook recipes depend on some OpsWorks Stacks built-in recipes. To make each recipe's origin clear, this topic identifies recipes using the Chef *cookbookname*::*recipename* convention.

Topics

- Attributes File
- Setup Recipes
- Configure Recipes
- Deploy Recipes
- Create a Stack and Run an Application

Attributes File

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Before looking at the recipes, it is useful to first examine the Tomcat cookbook's attributes file, which contains variety of configuration settings that the recipes use. Attributes aren't required; you can simply hardcode these values in your recipes or templates. However, if you define configuration settings using attributes, you can use the OpsWorks Stacks console or API to modify the values by defining custom JSON attributes, which is simpler and more flexible than rewriting the recipe or template code every time you want to change a setting. This approach allows you, for example, to use the same cookbook for multiple stacks, but configure the Tomcat server differently for each stack. For more information on attributes and how to override them, see Overriding Attributes.

The following example shows the complete attributes file, `default.rb`, which is located in the Tomcat cookbook's `attributes` directory.

```
default['tomcat']['base_version'] = 6
default['tomcat']['port'] = 8080
default['tomcat']['secure_port'] = 8443
default['tomcat']['ajp_port'] = 8009
default['tomcat']['shutdown_port'] = 8005
default['tomcat']['uri_encoding'] = 'UTF-8'
default['tomcat']['unpack_wars'] = true
default['tomcat']['auto_deploy'] = true
case node[:platform]
when 'centos', 'redhat', 'fedora', 'amazon'
  default['tomcat']['java_opts'] = ''
when 'debian', 'ubuntu'
  default['tomcat']['java_opts'] = '-Djava.awt.headless=true -Xmx128m -XX:
+UseConcMarkSweepGC'
end
default['tomcat']['catalina_base_dir'] = "/etc/tomcat#{node['tomcat']['base_version']}"
```

```
default['tomcat']['webapps_base_dir'] = "/var/lib/tomcat#{node['tomcat']
['base_version']}/webapps"
default['tomcat']['lib_dir'] = "/usr/share/tomcat#{node['tomcat']['base_version']}/lib"
default['tomcat']['java_dir'] = '/usr/share/java'
default['tomcat']['mysql_connector_jar'] = 'mysql-connector-java.jar'
default['tomcat']['apache_tomcat_bind_mod'] = 'proxy_http' # or: 'proxy_ajp'
default['tomcat']['apache_tomcat_bind_config'] = 'tomcat_bind.conf'
default['tomcat']['apache_tomcat_bind_path'] = '/tc/'
default['tomcat']['webapps_dir_entries_to_delete'] = %w(config log public tmp)
case node[:platform]
when 'centos', 'redhat', 'fedora', 'amazon'
  default['tomcat']['user'] = 'tomcat'
  default['tomcat']['group'] = 'tomcat'
  default['tomcat']['system_env_dir'] = '/etc/sysconfig'
when 'debian', 'ubuntu'
  default['tomcat']['user'] = "tomcat#{node['tomcat']['base_version']}"
  default['tomcat']['group'] = "tomcat#{node['tomcat']['base_version']}"
  default['tomcat']['system_env_dir'] = '/etc/default'
end
```

The settings themselves are discussed later in the related section. The following notes apply generally:

- All of the node definitions are `default` type, so you can override them with custom JSON attributes.
- The file uses a `case` statement to conditionally set some attribute values based on instance's operating system.

  The `platform` node is generated by Chef's Ohai tool and represents the instance's operating system.

Setup Recipes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Setup recipes are assigned to the layer's Setup lifecycle event and run after an instance boots. They perform tasks such as installing packages, creating configuration files, and starting services. After the Setup recipes finish running, OpsWorks Stacks runs the Deploy recipes to deploy any apps to the new instance.

Topics

- tomcat::setup
- tomcat::install
- tomcat::service
- tomcat::container_config
- tomcat::apache_tomcat_bind

tomcat::setup

The `tomcat::setup` recipe is intended to be assigned to a layer's Setup lifecycle event.

```
include_recipe 'tomcat::install'
include_recipe 'tomcat::service'

service 'tomcat' do
  action :enable
end

# for EBS-backed instances we rely on autofs
bash '(re-)start autofs earlier' do
  user 'root'
  code <<-EOC
    service autofs restart
  EOC
  notifies :restart, resources(:service => 'tomcat')
end

include_recipe 'tomcat::container_config'
include_recipe 'apache2'
include_recipe 'tomcat::apache_tomcat_bind'
```

`tomcat::setup` recipe is largely a metarecipe. It includes a set of dependent recipes that handle most of the details of installing and configuring Tomcat and related packages. The first part of `tomcat::setup` runs the following recipes, which are discussed later:

- The tomcat::install recipe installs the Tomcat server package.
- The tomcat::service recipe sets up the Tomcat service.

The middle part of `tomcat::setup` enables and starts the Tomcat service:

- The Chef service resource enables the Tomcat service at boot.
- The Chef bash resource runs a Bash script to start the autofs daemon, which is necessary for Amazon EBS-backed instances. The resource then notifies the `service` resource to restart the Tomcat service.

  For more information, see: autofs (for Amazon Linux) or Autofs (for Ubuntu).

The final part of `tomcat::setup` creates configuration files and installs and configures the front-end Apache server:

- The tomcat::container_config recipe creates configuration files.
- The `apache2` recipe (which is shorthand for `apache2::default`) is an OpsWorks Stacks built-in recipe that installs and configures an Apache server.
- The tomcat::apache_tomcat_bind recipe configures the Apache server to function as a front-end for the Tomcat server.

> **Note**
>
> You can often save time and effort by using built-in recipes to perform some of the required tasks. This recipe uses the built in `apache2::default` recipe to install Apache rather than implementing it from scratch. For another example of how to use built-in recipes, see Deploy Recipes.

The following sections describe the Tomcat cookbook's Setup recipes in more detail. For more information on the `apache2` recipes, see opsworks-cookbooks/apache2.

tomcat::install

The `tomcat::install` recipe installs the Tomcat server, the OpenJDK, and a Java connector library that handles the connection to the MySQL server.

```
tomcat_pkgs = value_for_platform(
  ['debian', 'ubuntu'] => {
    'default' => ["tomcat#{node['tomcat']['base_version']}", 'libtcnative-1',
 'libmysql-java']
  },
  ['centos', 'redhat', 'fedora', 'amazon'] => {
    'default' => ["tomcat#{node['tomcat']['base_version']}", 'tomcat-native', 'mysql-
connector-java']
  },
  'default' => ["tomcat#{node['tomcat']['base_version']}"]
)

tomcat_pkgs.each do |pkg|
  package pkg do
    action :install
  end
end

link ::File.join(node['tomcat']['lib_dir'], node['tomcat']['mysql_connector_jar']) do
  to ::File.join(node['tomcat']['java_dir'], node['tomcat']['mysql_connector_jar'])
  action :create
end

# remove the ROOT webapp, if it got installed by default
include_recipe 'tomcat::remove_root_webapp'
```

The recipe performs the following tasks:

1. Creates a list of packages to be installed, depending on the instance's operating system.
2. Installs each package in the list.

   The Chef package resource uses the appropriate provider—`yum` for Amazon Linux and `apt-get` for Ubuntu— to handle the installation. The package providers install OpenJDK as a Tomcat dependency, but the MySQL connector library must be installed explicitly.
3. Uses a Chef link resource to create a symlink in the Tomcat server's lib directory to the MySQL connector library in the JDK.

Using the default attribute values, the Tomcat lib directory is `/usr/share/tomcat6/lib` and the MySQL connector library (`mysql-connector-java.jar`) is in `/usr/share/java/`.

The `tomcat::remove_root_webapp` recipe removes the ROOT web application (`/var/lib/tomcat6/webapps/ROOT` by default) to avoid some security issues.

```
ruby_block 'remove the ROOT webapp' do
  block do
    ::FileUtils.rm_rf(::File.join(node['tomcat']['webapps_base_dir'], 'ROOT'), :secure
 => true)
  end
  only_if { ::File.exists?(::File.join(node['tomcat']['webapps_base_dir'], 'ROOT'))
 && !::File.symlink?(::File.join(node['tomcat']['webapps_base_dir'], 'ROOT')) }
end
```

The `only_if` statement ensures that the recipe removes the file only if it exists.

Note

The Tomcat version is specified by the `['tomcat']['base_version']` attribute, which is set to 6 in the attributes file. To install Tomcat 7, you can use custom JSON attributes to override the attribute. Just edit your stack settings and enter the following JSON in the **Custom Chef JSON** box, or add it to any existing custom JSON:

```
{
  'tomcat' : {
    'base_version' : 7
  }
}
```

The custom JSON attribute overrides the default attribute and sets the Tomcat version to 7. For more information on overriding attributes, see Overriding Attributes.

tomcat::service

The `tomcat::service` recipe creates the Tomcat service definition.

```
service 'tomcat' do
  service_name "tomcat#{node['tomcat']['base_version']}"

  case node[:platform]
  when 'centos', 'redhat', 'fedora', 'amazon'
    supports :restart => true, :reload => true, :status => true
  when 'debian', 'ubuntu'
    supports :restart => true, :reload => false, :status => true
  end

  action :nothing
end
```

The recipe uses the Chef service resource to specify the Tomcat service name (tomcat6, by default) and sets the `supports` attribute to define how Chef manages the service's restart, reload, and status commands on the different operating systems.

- `true` indicates that Chef can use the init script or other service provider to run the command
- `false` indicates that Chef must attempt to run the command by other means.

Notice that the `action` is set to `:nothing`. For each lifecycle event, OpsWorks Stacks initiates a Chef run to execute the appropriate set of recipes. The Tomcat cookbook follows a common pattern of having a recipe create the service definition, but not restart the service. Other recipes in the Chef run handle the restart, typically by including a `notifies` command in the `template` resources that are used to create configuration files. Notifications are a convenient way to restart a service because they do so only if the configuration has changed. In addition, if a Chef run has multiple restart notifications for a service, Chef restarts the service at most once. This practice avoids problems that can occur when attempting to restart a service that is not fully operational, which is a common source of Tomcat errors.

The Tomcat service must be defined for any Chef run that uses restart notifications. `tomcat::service` is therefore included in several recipes, to ensure that the service is defined for every Chef run. There is no penalty if a Chef run includes multiple instances of `tomcat::service` because Chef ensures that a recipe executes only once per run, regardless of how many times it is included.

tomcat::container_config

The `tomcat::container_config` recipe creates configuration files from cookbook template files.

```
include_recipe 'tomcat::service'

template 'tomcat environment configuration' do
  path ::File.join(node['tomcat']['system_env_dir'], "tomcat#{node['tomcat']
['base_version']}")
  source 'tomcat_env_config.erb'
  owner 'root'
  group 'root'
  mode 0644
  backup false
  notifies :restart, resources(:service => 'tomcat')
end

template 'tomcat server configuration' do
  path ::File.join(node['tomcat']['catalina_base_dir'], 'server.xml')
  source 'server.xml.erb'
  owner 'root'
  group 'root'
  mode 0644
  backup false
  notifies :restart, resources(:service => 'tomcat')
end
```

The recipe first calls `tomcat::service`, which defines the service if necessary. The bulk of the recipe consists of two template resources, each of which creates a configuration file from one of the cookbook's template files, sets the file properties, and notifies Chef to restart the service.

Tomcat Environment Configuration File

The first `template` resource uses the `tomcat_env_config.erb` template file to create a Tomcat environment configuration file, which is used to set environment variables such as `JAVA_HOME`. The default file name is the `template` resource's argument. `tomcat::container_config` uses a `path` attribute to override the default value and name the configuration file `/etc/sysconfig/tomcat6` (Amazon Linux) or `/etc/default/tomcat6` (Ubuntu). The `template` resource also specifies the file's owner, group, and mode settings and directs Chef to not create backup files.

If you look at the source code, there are actually three versions of `tomcat_env_config.erb`, each in a different subdirectory of the `templates` directory. The `ubuntu` and `amazon` directories contain the templates for their respective operating systems. The `default` folder contains a dummy template with a single comment line, which is used only if you attempt to run this recipe on an instance with an unsupported operating system. The `tomcat::container_config` recipe doesn't need to specify which `tomcat_env_config.erb` to use. Chef automatically picks the appropriate directory for the instance's operating system based on rules described in File Specificity.

The `tomcat_env_config.erb` files for this example consist largely of comments. To set additional environment variables, just uncomment the appropriate lines and provide your preferred values.

> **Note**
>
> Any configuration setting that might change should be defined as an attribute rather than hardcoded in the template. That way, you don't have to rewrite the template to change a setting, you can just override the attribute.

The Amazon Linux template sets only one environment variable, as shown in the following excerpt.

```
...
# Use JAVA_OPTS to set java.library.path for libtcnative.so
#JAVA_OPTS="-Djava.library.path=/usr/lib"

JAVA_OPTS="${JAVA_OPTS} <%= node['tomcat']['java_opts'] %>"

# What user should run tomcat
#TOMCAT_USER="tomcat"
...
```

JAVA_OPTS can be used to specify Java options such as the library path. Using the default attribute values, the template sets no Java options for Amazon Linux. You can set your own Java options by overriding the `['tomcat']['java_opts']` attribute, for example, by using custom JSON attributes. For an example, see Create a Stack.

The Ubuntu template sets several environment variables, as shown in the following template excerpt.

```
# Run Tomcat as this user ID. Not setting this or leaving it blank will use the
# default of tomcat<%= node['tomcat']['base_version'] %>.
TOMCAT<%= node['tomcat']['base_version'] %>_USER=tomcat<%= node['tomcat']
['base_version'] %>
...
# Run Tomcat as this group ID. Not setting this or leaving it blank will use
# the default of tomcat<%= node['tomcat']['base_version'] %>.
TOMCAT<%= node['tomcat']['base_version'] %>_GROUP=tomcat<%= node['tomcat']
['base_version'] %>
...
JAVA_OPTS="<%= node['tomcat']['java_opts'] %>"

<% if node['tomcat']['base_version'].to_i < 7 -%>
# Unset LC_ALL to prevent user environment executing the init script from
# influencing servlet behavior.  See Debian bug #645221
unset LC_ALL
<% end -%>
```

Using default attribute values, the template sets the Ubuntu environment variables as follows:

- `TOMCAT6_USER` and `TOMCAT6_GROUP`, which represent the Tomcat user and group, are both set to `tomcat6`.

  If you set ['tomcat']['base_version'] to `tomcat7`, the variable names resolve to `TOMCAT7_USER` and `TOMCAT7_GROUP`, and both are set to `tomcat7`.
- `JAVA_OPTS` is set to `-Djava.awt.headless=true -Xmx128m -XX:+UseConcMarkSweepGC`:
  - Setting `-Djava.awt.headless` to `true` informs the graphics engine that the instance is headless and does not have a console, which addresses faulty behavior of certain graphical applications.
  - `-Xmx128m` ensures that the JVM has adequate memory resources, 128MB for this example.
  - `-XX:+UseConcMarkSweepGC` specifies concurrent mark sweep garbage collection, which helps limit garbage-collection induced pauses.

    For more information, see: Concurrent Mark Sweep Collector Enhancements.
- If the Tomcat version is less than 7, the template unsets `LC_ALL`, which addresses a Ubuntu bug.

> **Note**
>
> With the default attributes, some of these environment variables are simply set to their default values. However, explicitly setting environment variables to attributes means you can define custom JSON attributes to override the default attributes and provide custom values. For more information on overriding attributes, see Overriding Attributes.

For the complete template files, see the source code.

Server.xml Configuration File

The second `template` resource uses `server.xml.erb` to create the `system.xml` configuration file, which configures the servlet/JSP container. `server.xml.erb` contains no operating system-specific settings, so it is in the `template` directory's `default` subdirectory.

The template uses standard settings, but it can create a `system.xml` file for either Tomcat 6 or Tomcat 7. For example, the following code from the template's server section configures the listeners appropriately for the specified version.

```
<% if node['tomcat']['base_version'].to_i > 6 -%>
  <!-- Security listener. Documentation at /docs/config/listeners.html
  <Listener className="org.apache.catalina.security.SecurityListener" />
  -->
<% end -%>
  <!--APR library loader. Documentation at /docs/apr.html -->
  <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
  <!--Initialize Jasper prior to webapps are loaded. Documentation at /docs/jasper-
howto.html -->
  <Listener className="org.apache.catalina.core.JasperListener" />
  <!-- Prevent memory leaks due to use of particular java/javax APIs-->
  <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
<% if node['tomcat']['base_version'].to_i < 7 -%>
  <!-- JMX Support for the Tomcat server. Documentation at /docs/non-existent.html -->
  <Listener className="org.apache.catalina.mbeans.ServerLifecycleListener" />
<% end -%>
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
<% if node['tomcat']['base_version'].to_i > 6 -%>
  <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />
<% end -%>
```

The template uses attributes in place of hardcoded settings so you can easily change the settings by defining custom JSON attributes. For example:

```
<Connector port="<%= node['tomcat']['port'] %>" protocol="HTTP/1.1"
           connectionTimeout="20000"
           URIEncoding="<%= node['tomcat']['uri_encoding'] %>"
           redirectPort="<%= node['tomcat']['secure_port'] %>" />
```

For more information, see the source code.

tomcat::apache_tomcat_bind

The `tomcat::apache_tomcat_bind` recipe enables the Apache server to act as Tomcat's front end, receiving incoming requests and forwarding them to Tomcat and returning the responses to the client. This example uses mod_proxy as the Apache proxy/gateway.

```
execute 'enable mod_proxy for apache-tomcat binding' do
  command '/usr/sbin/a2enmod proxy'
  not_if do
    ::File.symlink?(::File.join(node['apache']['dir'], 'mods-enabled', 'proxy.load'))
 || node['tomcat']['apache_tomcat_bind_mod'] !~ /\Aproxy/
  end
end

execute 'enable module for apache-tomcat binding' do
  command "/usr/sbin/a2enmod #{node['tomcat']['apache_tomcat_bind_mod']}"
  not_if {::File.symlink?(::File.join(node['apache']['dir'], 'mods-enabled',
 "#{node['tomcat']['apache_tomcat_bind_mod']}.load"))}
end

include_recipe 'apache2::service'

template 'tomcat thru apache binding' do
  path ::File.join(node['apache']['dir'], 'conf.d', node['tomcat']
['apache_tomcat_bind_config'])
  source 'apache_tomcat_bind.conf.erb'
  owner 'root'
  group 'root'
  mode 0644
  backup false
  notifies :restart, resources(:service => 'apache2')
```

```
end
```

To enable mod_proxy, you must enable the proxy module and a protocol-based module. You have two options for the protocol module:

- HTTP: proxy_http
- Apache JServ Protocol (AJP): proxy_ajp

  AJP is an internal Tomcat protocol.

Both the recipe's execute resources run the a2enmod command, which enables the specified module by creating the required symlinks:

- The first execute resource enables the proxy module.
- The second execute resource enables the protocol module, which is set to proxy_http by default.

  If you would rather use AJP, you can define custom JSON to override the apache_tomcat_bind_mod attribute and set it to proxy_ajp.

The apache2::service recipe is an OpsWorks Stacks built-in recipe that defines the Apache service. For more information, see the recipe in the OpsWorks Stacks GitHub repository.

The template resource uses apache_tomcat_bind.conf.erb to create a configuration file, named tomcat_bind.conf by default. It places the file in the ['apache']['dir']/.conf.d directory. The ['apache']['dir'] attribute is defined in the built-in apache2 attributes file, and is set by default to /etc/httpd (Amazon Linux), or /etc/apache2 (Ubuntu). If the template resource creates or changes the configuration file, the notifies command schedules an Apache service restart.

```
<% if node['tomcat']['apache_tomcat_bind_mod'] == 'proxy_ajp' -%>
ProxyPass <%= node['tomcat']['apache_tomcat_bind_path'] %> ajp://localhost:<%=
 node['tomcat']['ajp_port'] %>/
ProxyPassReverse <%= node['tomcat']['apache_tomcat_bind_path'] %> ajp://localhost:<%=
 node['tomcat']['ajp_port'] %>/
<% else %>
ProxyPass <%= node['tomcat']['apache_tomcat_bind_path'] %> http://localhost:<%=
 node['tomcat']['port'] %>/
```

```
ProxyPassReverse <%= node['tomcat']['apache_tomcat_bind_path'] %> http://localhost:<%=
 node['tomcat']['port'] %>/
<% end -%>
```

The template uses the ProxyPass and ProxyPassReverse directives to configure the port used to pass traffic between Apache and Tomcat. Because both servers are on the same instance, they can use a localhost URL and are both set by default to `http://localhost:8080`.

Configure Recipes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Configure recipes are assigned to the layer's Configure lifecycle event, which occurs on all of the stack's instances whenever an instance enters or leaves the online state. You use Configure recipes to adjust an instance's configuration to respond to the change, as appropriate. When you implement a Configure recipe, keep in mind that a stack configuration change might involve instances that have nothing to do with this layer. The recipe must be able to respond appropriately, which might mean doing nothing in some cases.

tomcat::configure

The `tomcat::configure` recipe is intended for a layer's Configure lifecycle event.

```
include_recipe 'tomcat::context'
# Optional: Trigger a Tomcat restart in case of a configure event, if relevant
# settings in custom JSON have changed (e.g. java_opts/JAVA_OPTS):
#include_recipe 'tomcat::container_config'
```

The `tomcat::configure` recipe is basically a metarecipe that runs two dependent recipes.

1. The `tomcat::context` recipe create a web app context configuration file.

This file configures the JDBC resources that applications use to communicate with the MySQL instance, as discussed in the next section. Running this recipe in response to a configure event allows the layer to update the web app context configuration file if the database layer has changed.

2. The `tomcat::container_config` Setup recipe is run again to capture any changes in the container configuration.

The `include` for `tomcat::container_config` is commented out for this example. If you want to use custom JSON to modify Tomcat settings, you can remove the comment. A Configure lifecycle event then runs `tomcat::container_config`, which updates the Tomcat related configuration files, as described in tomcat::container_config and restarts the Tomcat service.

tomcat::context

The Tomcat cookbook enables applications to access a MySQL database server, which can be running on a separate instance, by using a J2EE DataSource object. With Tomcat, you can enable the connection by creating and installing a web app context configuration file for each application. This file defines the relationship between the application and the JDBC resource that the application will use to communicate with the database. For more information, see The Context Container.

The `tomcat::context` recipe's primary purpose is to create this configuration file.

```
include_recipe 'tomcat::service'

node[:deploy].each do |application, deploy|
  context_name = deploy[:document_root].blank? ? application : deploy[:document_root]

  template "context file for #{application} (context name: #{context_name})" do
    path ::File.join(node['tomcat']['catalina_base_dir'], 'Catalina', 'localhost',
 "#{context_name}.xml")
    source 'webapp_context.xml.erb'
    owner node['tomcat']['user']
    group node['tomcat']['group']
    mode 0640
    backup false
    only_if { node['datasources'][context_name] }
    variables(:resource_name => node['datasources'][context_name], :webapp_name =>
 application)
```

```
    notifies :restart, resources(:service => 'tomcat')
  end
end
```

In addition to Tomcat cookbook attributes, this recipe uses the stack configuration and deployment attributes that OpsWorks Stacks installs with the Configure event. The OpsWorks Stacks service adds attributes to each instance's node object that contain the information that recipes would typically obtain by using data bags or search and installs the attributes on each instance. The attributes contain detailed information about the stack configuration, deployed apps, and any custom data that a user wants to include. Recipes can obtain data from stack configuration and deployment attributes by using standard Chef node syntax. For more information, see Stack Configuration and Deployment Attributes. With Chef 11.10 stacks, you also can use Chef search to obtain stack configuration and deployment data. For more information, see Using Chef Search.

`deploy` attributes refers to the `[:deploy]` namespace, which contains deployment-related attributes that are defined through the console or API, or generated by the OpsWorks Stacks service. The `deploy` attribute includes an attribute for each deployed app, named with the app's short name. Each app attribute contains a set of attributes that characterize the app, such as the document root (`[:deploy][:`*`appname`*`][:document_root]`).

The `context` recipe first ensures that the service is defined for this Chef run by calling tomcat::service. It then defines a `context_name` variable which represents the configuration file's name, excluding the `.xml` extension. If you use the default document root, `context_name` is set to the app's short name. Otherwise, it is set to the specified document root. The example discussed in Create a Stack and Run an Application sets the document root to `"ROOT"`, so the context is ROOT and the configuration file is named `ROOT.xml`.

The bulk of the recipe goes through the list of deployed apps and for each app, uses the `webapp_context.xml.erb` template to create a context configuration file. The example deploys only one app, but the definition of the `deploy` attribute requires you to treat it as a list of apps regardless.

The `webapp_context.xml.erb` template is not operating-system specific, so it is located in the `templates` directory's `default` subdirectory.

The recipe creates the configuration file as follows:

- Using default attribute values, the configuration file name is set to *`context_name`*`.xml` and installed in the `/etc/tomcat6/Catalina/localhost/` directory.

The `['datasources']` node from the stack configuration attributes contains one or more attributes, each of which maps a context name to the JDBC data resource that the associated application will use to communicate with the database. The node and its contents are defined with custom JSON when you create the stack, as described later in Create a Stack and Run an Application. The example has a single attribute that associates the ROOT context name with a JDBC resource named jdbc/mydb.

- Using default attribute values, the file's user and group are both set to the values defined by the Tomcat package: `tomcat` (Amazon Linux) or `tomcat6` (Ubuntu).
- The `template` resource creates the configuration file only if the `['datasources']` node exists and includes a `context_name` attribute.
- The `template` resource defines two variables, `resource_name` and `webapp_name`.

  `resource_name` is set to the resource name that is associated with `context_name` and `webapp_name` is set to the app's short name.
- The template resource restarts the Tomcat service to load and activate the changes.

The `webapp_context.xml.erb` template consists of a `Context` element that contains a `Resource` element with its own set of attributes.

The `Resource` attributes characterize the context configuration:

- **name**–The JDBC resource name, which is set to the `resource_name` value defined in `tomcat::context`.

  For the example, the resource name is set to jdbc/mydb.
- **auth** and **type**–These are standard settings for JDBC `DataSource` connections.
- **maxActive**, **maxIdle**, and **maxWait**–The maximum number of active and idle connections, and the maximum wait time for a connection to be returned.
- **username**, and **password**–The database's user name and root password, which are obtained from the `deploy` attributes.
- **driverClassName**–The JDBC driver's class name, which is set to the MySQL driver.
- **url**–The connection URL.

  The prefix depends on the database. It should be set to `jdbc:mysql` for MySQL, `jdbc:postgresql` for Postgres, and `jdbc:sqlserver` for SQL Server. The example sets the

URL to `jdbc:mysql://host_IP_Address:3306:simplejsp`, where *`simplejsp`* is the app's short name.

- **factory**–The `DataSource` factory, which is required for MySQL databases.

For more information on this configuration file, see the Tomcat wiki's Using DataSources topic.

Deploy Recipes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Deploy recipes are assigned to the layer's Deploy lifecycle event. It typically occurs on all of the stack's instances whenever you deploy an app, although you can optionally restrict the event to only specified instances. OpsWorks Stacks also runs the Deploy recipes on new instances, after the Setup recipes complete. The primary purpose of Deploy recipes is to deploy code and related files from a repository to the application server layer's instances. However, you often run Deploy recipes on other layers as well. This allows those layers' instances, for example, to update their configuration to accommodate the newly deployed app. When you implement a Deploy recipe, keep in mind a Deploy event does not necessarily mean that apps are being deployed to the instance. It could simply be a notification that apps are being deployed to other instances in the stack, to allow the instance to make any necessary updates. The recipe must be able to respond appropriately, which might mean doing nothing.

OpsWorks Stacks automatically deploys apps of the standard app types to the corresponding built-in application server layers. To deploy apps to a custom layer, you must implement custom Deploy recipes that download the app's files from a repository to the appropriate location on the instance. However, you can often limit the amount of code you must write by using the built-in deploy cookbook to handle some aspects of deployment. For example, if you store your files in one of the supported repositories, the built-in cookbook can handle the details of downloading the files from the repository to the layer's instances.

The `tomcat::deploy` recipe is intended to be assigned to the Deploy lifecycle event.

```
include_recipe 'deploy'

node[:deploy].each do |application, deploy|
  opsworks_deploy_dir do
    user deploy[:user]
    group deploy[:group]
    path deploy[:deploy_to]
  end

  opsworks_deploy do
    deploy_data deploy
    app application
  end
...
```

The `tomcat::deploy` recipe uses the built-in deploy cookbook for aspects of deployment that aren't application specific. The `deploy` recipe (which is shorthand for the built-in `deploy::default` recipe) is a built-in recipe that handles the details of setting up the users, groups, and so on, based on data from the `deploy` attributes.

The recipe uses two built-in Chef definitions, `opsworks_deploy_dir` and `opworks_deploy` to install the application.

The `opsworks_deploy_dir` definition sets up the directory structure, based on data from the app's deployment JSON. Definitions are basically a convenient way to package resource definitions, and are located in a cookbook's `definitions` directory. Recipes can use definitions much like resources, but the definition itself does not have an associated provider, just the resources that are included in the definition. You can define variables in the recipe, which are passed to the underlying resource definitions. The `tomcat::deploy` recipe sets `user`, `group`, and `path` variables based on data from the deployment JSON. They are passed to the definition's directory resource, which manages the directories.

Note

Your deployed app's user and group are determined by the `[:opsworks][:deploy_user][:user]` and `[:opsworks][:deploy_user][:group]` attributes, which are defined in the built-in deploy cookbook's `deploy.rb` attributes file. The default value of `[:opsworks][:deploy_user][:user]` is `deploy`. The default value of `[:opsworks][:deploy_user][:group]` depends on the instance's operating system:

- For Ubuntu instances, the default group is `www-data`.
- For Amazon Linux instances that are members of a Rails App Server layer that uses Nginx and Unicorn, the default group is `nginx`.
- For all other Amazon Linux instances, the default group is `apache`.

You can change either setting by using custom JSON or a custom attributes file to override the appropriate attribute. For more information, see Overriding Attributes.

The other definition, `opsworks_deploy`, handles the details of checking out the app's code and related files from the repository and deploying them to the instance, based on data from the `deploy` attributes. You can use this definition for any app type; deployment details such as the directory names are specified in the console or through the API and put in the `deploy` attributes. However, `opsworks_deploy` works only for the four supported repository types: Git, Subversion, S3, and HTTP. You must implement this code yourself if you want to use a different repository type.

You install an app's files in the Tomcat `webapps` directory. A typical practice is to copy the files directly to `webapps`. However, OpsWorks Stacks deployment is designed to retain up to five versions of an app on an instance, so you can roll back to an earlier version if necessary. OpsWorks Stacks therefore does the following:

1. Deploys apps to a distinct directory whose name contains a time stamp, such as `/srv/www/my_1st_jsp/releases/20130731141527`.
2. Creates a symlink named `current`, such as `/srv/www/my_1st_jsp/current`, to this unique directory.
3. If does not already exist, creates a symlink from the `webapps` directory to the `current` symlink created in Step 2.

If you need to roll back to an earlier version, modify the `current` symlink to point to a distinct directory containing the appropriate timestamp, for example, by changing the link target of `/srv/www/my_1st_jsp/current`.

The middle section of `tomcat::deploy` sets up the symlink.

```
  ...
  current_dir = ::File.join(deploy[:deploy_to], 'current')
  webapp_dir = ::File.join(node['tomcat']['webapps_base_dir'],
 deploy[:document_root].blank? ? application : deploy[:document_root])

  # opsworks_deploy creates some stub dirs, which are not needed for typical webapps
  ruby_block "remove unnecessary directory entries in #{current_dir}" do
    block do
      node['tomcat']['webapps_dir_entries_to_delete'].each do |dir_entry|
        ::FileUtils.rm_rf(::File.join(current_dir, dir_entry), :secure => true)
      end
    end
  end

  link webapp_dir do
    to current_dir
    action :create
  end
  ...
```

The recipe first creates two variables, `current_dir` and `webapp_dir` to represent the `current` and `webapp` directories, respectively. It then uses a `link` resource to link `webapp_dir` to `current_dir`. The OpsWorks Stacks `deploy::default` recipe creates some stub directories that aren't required for this example, so the middle part of the excerpt removes them.

The final part of `tomcat::deploy` restarts the Tomcat service, if necessary.

```
  ...
  include_recipe 'tomcat::service'

  execute 'trigger tomcat service restart' do
    command '/bin/true'
    not_if { node['tomcat']['auto_deploy'].to_s == 'true' }
    notifies :restart, resources(:service => 'tomcat')
  end
end

include_recipe 'tomcat::context'
```

The recipe first runs `tomcat::service`, to ensure that the service is defined for this Chef run. It then uses an execute resource to notify the service to restart, but only if `['tomcat']`

`['auto_deploy']` is set to `'true'`. Otherwise, Tomcat listens for changes in its `webapps` directory, which makes an explicit Tomcat service restart unnecessary.

> **Note**
>
> The `execute` resource doesn't actually execute anything substantive; `/bin/true` is a dummy shell script that simply returns a success code. It is used here simply as a convenient way to generate a restart notification. As mentioned earlier, using notifications ensures that services are not restarted too frequently.

Finally, `tomcat::deploy` runs `tomcat::context`, which updates the web app context configuration file if you have changed the back end database.

Create a Stack and Run an Application

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section shows how to use the Tomcat cookbook to implement a basic stack setup that runs a simple Java server pages (JSP) application named SimpleJSP. The stack consists of a Tomcat-based custom layer named TomCustom and a MySQL layer. SimpleJSP is deployed to TomCustom and displays some information from the MySQL database. If you are not already familiar with the basics of how to use OpsWorks Stacks, you should first read Getting Started with Chef 11 Linux Stacks.

The SimpleJSP Application

The SimpleJSP application demonstrates the basics of how to set up a database connection and retrieve data from the stack's MySQL database.

```
<html>
  <head>
    <title>DB Access</title>
```

```
  </head>
  <body>
    <%@ page language="java" import="java.sql.*,javax.naming.*,javax.sql.*" %>
    <%
      StringBuffer output = new StringBuffer();
      DataSource ds = null;
      Connection con = null;
      Statement stmt = null;
      ResultSet rs = null;
      try {
        Context initCtx = new InitialContext();
        ds = (DataSource) initCtx.lookup("java:comp/env/jdbc/mydb");
        con = ds.getConnection();
        output.append("Databases found:<br>");
        stmt = con.createStatement();
        rs = stmt.executeQuery("show databases");
        while (rs.next()) {
          output.append(rs.getString(1));
          output.append("<br>");
        }
      }
      catch (Exception e) {
        output.append("Exception: ");
        output.append(e.getMessage());
        output.append("<br>");
      }
      finally {
        try {
          if (rs != null) {
            rs.close();
          }
          if (stmt != null) {
            stmt.close();
          }
          if (con != null) {
            con.close();
          }
        }
        catch (Exception e) {
          output.append("Exception (during close of connection): ");
          output.append(e.getMessage());
          output.append("<br>");
        }
      }
```

```
    %>
    <%= output.toString() %>
  </body>
</html>
```

SimpleJSP uses a `DataSource` object to communicate with the MySQL database. Tomcat uses the data in the web app context configuration file to create and initialize a `DataSource` object and bind it to a logical name. It then registers the logical name with a Java Naming and Directory Interface (JNDI) naming service. To get an instance of the appropriate `DataSource` object, you create an `InitialContext` object and pass the resource's logical name to the object's `lookup` method, which retrieves the appropriate object. The SimpleJSP example's logical name, `java:comp/env/jdbc/mydb`, has the following components:

- The root namespace, `java`, which is separated from the rest of the name by a colon (:).
- Any additional namespaces, separated by forward slashes (/).

  Tomcat automatically adds resources to the `comp/env` namespace.
- The resource name, which is defined in the web app context configuration file and separated from the namespaces by a forward slash.

  The resource name for this example is `jdbc/mydb`.

To establish a connection to the database, SimpleJSP does the following:

1. Calls the `DataSource` object's `getConnection` method, which returns a `Connection` object.
2. Calls the `Connection` object's `createStatement` method to create a `Statement` object, which you use to communicate with the database.
3. Communicates with the database by calling the appropriate `Statement` method.

   SimpleJSP calls `executeQuery` to execute a SHOW DATABASES query, which lists the server's databases.

The `executeQuery` method returns a `ResultSet` object, which contains the query results. SimpleJSP gets the database names from the returned `ResultSet` object and concatenates them to create an output string. Finally, the example closes the `ResultSet`, `Statement`, and

`Connection` objects. For more information about JSP and JDBC, see JavaServer Pages Technology and JDBC Basics, respectively.

To use SimpleJSP with a stack, you must put it in a repository. You can use any of the supported repositories, but to use SimpleJSP with the example stack discussed in the following section, you must put it in a public S3 archive. For information on how to use the other standard repositories, see Cookbook Repositories.

To put SimpleJSP in an S3 archive repository

1. Copy the example code to a file named `simplejsp.jsp` and put the file in a directory named `simplejsp`.
2. Create a `.zip` archive of the `simplejsp` directory.
3. Create a public Amazon S3 bucket, upload `simplejsp.zip` to the bucket, and make the file public.

   For a description of how to perform this task, see Get Started With Amazon Simple Storage Service.

Create a Stack

To run SimpleJSP you need a stack with the following layers.

- A MySQL layer, that supports the back end MySQL server.
- A custom layer that uses the Tomcat cookbook to support Tomcat server instances.

To create the stack

1. On the OpsWorks Stacks dashboard, click **Add Stack** to create a new stack and click **Advanced >>** to display all options. Configure the stack as follows.

   - **Name**–A user-defined stack name; this example uses TomStack.
   - **Use custom Chef cookbooks**–Set the toggle to **Yes**, which displays some additional options.
   - **Repository type**–Git.
   - **Repository URL**–`git://github.com/amazonwebservices/opsworks-example-cookbooks.git`.
   - **Custom Chef JSON**–Add the following JSON:

```
{
  "tomcat": {
    "base_version": 7,
    "java_opts": "-Djava.awt.headless=true -Xmx256m"
  },
  "datasources": {
    "ROOT": "jdbc/mydb"
  }
}
```

For the remaining options, you can accept the defaults.

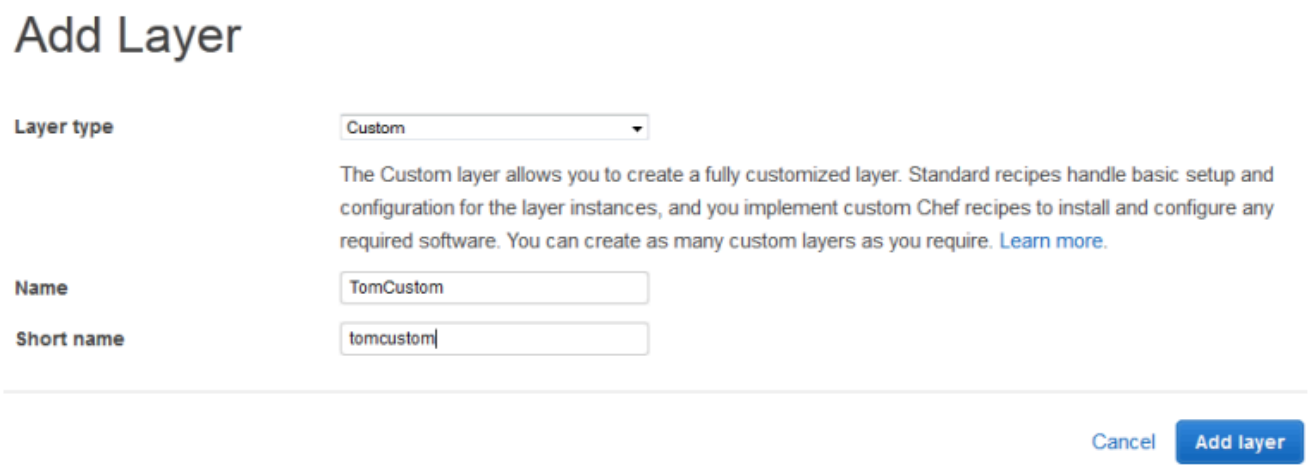
The custom JSON does the following:

- Overrides the Tomcat cookbook's `['base_version']` attribute to set the Tomcat version to 7; the default value is 6.
- Overrides the Tomcat cookbook's `['java_opts']` attribute to specify that the instance is headless and set the JVM maximum heap size to 256MB; the default value sets no options for instances running Amazon Linux.
- Specifies the `['datasources]` attribute value, which assigns a JDBC resource name (jdbc/mydb) to the web app context name (ROOT), as discussed in tomcat::context.

  This last attribute has no default value; you must set it with custom JSON.



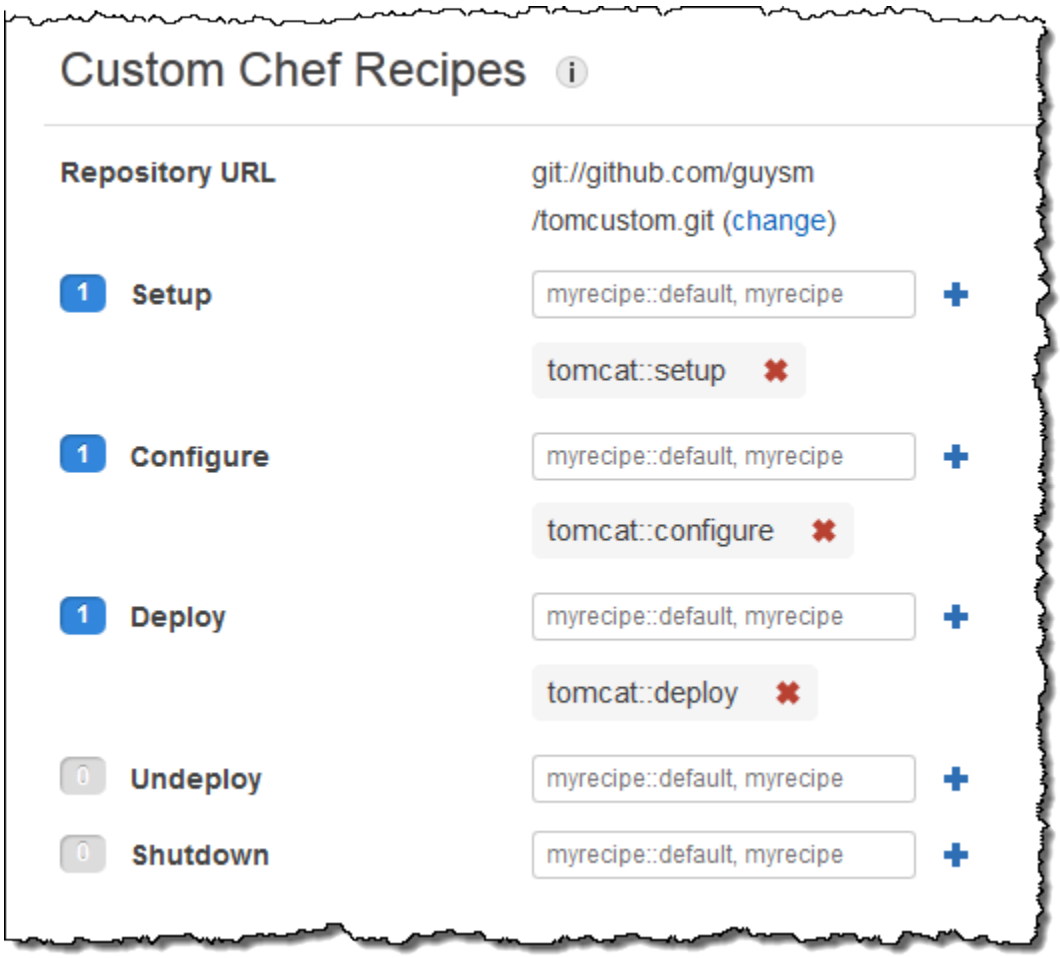
2. Click **Add a layer**. For **Layer type**, select **MySQL**. Then click **Add Layer**.
3. Click **Instances** in the navigation pane and then click **Add an instance**. Click **Add Instance** to accept the defaults. On the line for the instance, click **start**.
4. Return to the **Layers** page and click **+ Layer** to add a layer. For **Layer type**, click **Custom**.s The example uses **`TomCustom`** and **`tomcustom`** as the layer's name and short name, respectively.



5. On the **Layers** page, for the custom layer, click **Recipes** and then click **Edit**. Under **Custom Chef Recipes**, assign Tomcat cookbook recipes to the layer's lifecycle events, as follows:

   - For **Setup**, type **`tomcat::setup`** and click **+**.
   - For **Configure**, type **`tomcat::configure`** and click **+**.
   - For **Deploy**, type **`tomcat::deploy`** and click **+**. Then click **Save**.

   .



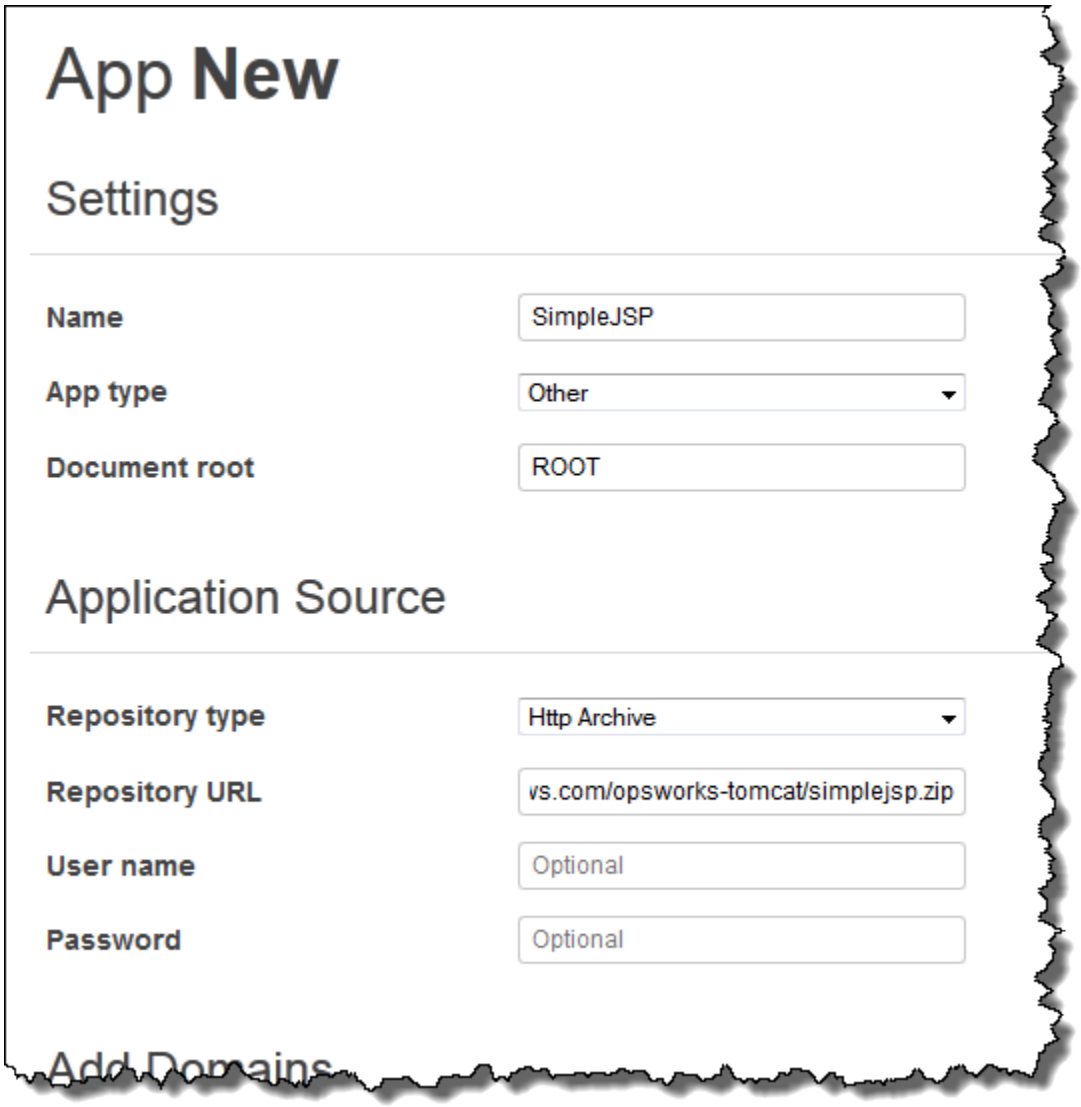
6. Click **Apps** in the navigation pane and then click **Add an app**. Specify the following options and then click **Add App**:

   - **Name**–The app's name; the example uses SimpleJSP and the short name generated by OpsWorks Stacks will be simplejsp.
   - **App type**–Set this option to **Other**.

     OpsWorks Stacks automatically deploys standard app types to the associated server instances. If you set **App type** to other, OpsWorks Stacks simply runs the Deploy recipes, and lets them handle deployment.
   - **Document root**–Set this option to **R00T**.

     The **Document root** value specifies the context name.
   - **Repository type**–Set this option to **S3 Archive**.
   - **Repository URL**–Set this to the app's Amazon S3 URL that you created earlier.

Use default settings for the other options.



7. Use the **Instances** page to add an instance to the TomCustom layer and start it. OpsWorks Stacks automatically runs the Deploy recipes on a new instance after the Setup recipes complete, so starting the instance also deploys SimpleJSP.
8. When the TomCustom instance is online, click the instance name on the **Instances** page to see its details. Copy the public IP address. Then construct a URL as follows: http://*publicIP*/tc/*appname.jsp*. For the example, this URL will look something like **`http://50.218.191.172/tc/simplejsp.jsp`**.

> **Note**
>
> The Apache URL that forwards requests to Tomcat is set to the default `['tomcat']['apache_tomcat_bind_path']` attribute, `/tc/`. The SimpleJSP document root is set to ROOT which is a special value that resolves to `/`. The URL is therefore ".../tc/simplejsp.jsp".

9. Paste the URL from the previous step into your browser. You should see the following:

```
Databases found:
information_schema
simplejsp
test
```

> **Note**
>
> If your stack has a MySQL instance, OpsWorks Stacks automatically creates a database for each app, named with the app's short name.

Stack Configuration and Deployment Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

When OpsWorks Stacks runs a command on an instance—for example, a deploy command in response to a Deploy lifecycle event—it adds a set of attributes to the instance's node object that describes the stack's current configuration. For Deploy events and Execute Recipes stack commands, OpsWorks Stacks installs deploy attributes, which provide some additional deployment information. For more information about the node object, see Overriding Attributes. For a list

of commonly used stack configuration and deployment attributes, including fully qualified node names, see Stack Configuration and Deployment Attributes: Linux and Built-in Cookbook Attributes.

> **Note**
>
> On Linux stacks, you can get a complete list of these attributes, formatted as a JSON object, by using the agent CLI's get_json command.

The following sections show the attributes associated with a Configure event and a Deploy event for a simple stack, which consists of the following:

- A PHP App Server layer with two instances
- An HAProxy layer with one instance

The examples are from one of the PHP App Server instances, **php-app1**. For convenience, the attributes are formatted as a JSON object. The object's structure maps to the attributes' fully qualified names. For example, the `node[:opsworks][:ruby_version]` attribute appears as follows in a JSON representation.

```
{
  "opsworks": {
    ...
    "ruby_version": "1.8.7",
    ...
  }
}
```

Topics

- Configure Attributes
- Deployment Attributes

Configure Attributes

The following JSON object shows the attributes for a Configure event, which occurs on every instance in the stack when an instance comes online or goes offline. The attributes include the

built-in stack configuration attributes and any custom JSON attributes that were defined for the stack prior to the event (none in this example). It has been edited for length. For a detailed description of the various attributes, see Stack Configuration and Deployment Attributes: Linux and Built-in Cookbook Attributes.

```
{
  "opsworks": {
    "layers": {
      "php-app": {
        "id": "4a2a56c8-f909-4b39-81f8-556536d20648",
        "instances": {
          "php-app2": {
            "elastic_ip": null,
            "region": "us-west-2",
            "booted_at": "2013-02-26T20:41:10+00:00",
            "ip": "192.0.2.0",
            "aws_instance_id": "i-34037f06",
            "availability_zone": "us-west-2a",
            "instance_type": "c1.medium",
            "private_dns_name": "ip-10-252-0-203.us-west-2.compute.internal",
            "private_ip": "10.252.0.203",
            "created_at": "2013-02-26T20:39:39+00:00",
            "status": "online",
            "backends": 8,
            "public_dns_name": "ec2-192-0-2-0.us-west-2.compute.amazonaws.com"
          },
          "php-app1": {
            ...
          }
        },
        "name": "PHP Application Server"
      },
      "lb": {
        "id": "15c86142-d836-4191-860f-f4d310440f14",
        "instances": {
          "lb1": {
           ...
          }
        },
        "name": "Load Balancer"
      }
    },
```

```
    "agent_version": "104",
    "applications": [

    ],
    "stack": {
      "name": "MyStack"
    },
    "ruby_version": "1.8.7",
    "sent_at": 1361911623,
    "ruby_stack": "ruby_enterprise",
    "instance": {
      "layers": [
        "php-app"
      ],
      "region": "us-west-2",
      "ip": "192.0.2.0",
      "id": "45ef378d-b87c-42be-a1b9-b67c48edafd4",
      "aws_instance_id": "i-32037f00",
      "availability_zone": "us-west-2a",
      "private_dns_name": "ip-10-252-84-253.us-west-2.compute.internal",
      "instance_type": "c1.medium",
      "hostname": "php-app1",
      "private_ip": "10.252.84.253",
      "backends": 8,
      "architecture": "i386",
      "public_dns_name": "ec2-192-0-2-0.us-west-2.compute.amazonaws.com"
    },
    "activity": "configure",
    "rails_stack": {
      "name": null
    },
    "deployment": null,
    "valid_client_activities": [
      "reboot",
      "stop",
      "setup",
      "configure",
      "update_dependencies",
      "install_dependencies",
      "update_custom_cookbooks",
      "execute_recipes"
    ]
  },
  "opsworks_custom_cookbooks": {
```

```
    "recipes": [

    ],
    "enabled": false
  },
  "recipes": [
    "opsworks_custom_cookbooks::load",
    "opsworks_ganglia::configure-client",
    "ssh_users",
    "agent_version",
    "mod_php5_apache2::php",
    "php::configure",
    "opsworks_stack_state_sync",
    "opsworks_custom_cookbooks::execute",
    "test_suite",
    "opsworks_cleanup"
  ],
  "opsworks_rubygems": {
    "version": "1.8.24"
  },
  "ssh_users": {
  },
  "opsworks_bundler": {
    "manage_package": null,
    "version": "1.0.10"
  },
  "deploy": {
  }
}
```

Most of the information is under the `opsworks` attribute, which is often referred to as a namespace. The following list describes the key attributes:

- `layers` attributes – A set of attributes, each of which describes the configuration of one of the stack's layers.

  The layers are identified by their shortnames, `php-app` and `lb` for this example. For more information about shortnames for other layers, see [OpsWorks Stacks Layer Reference].

- `instances` attributes – Every layer has an `instances` element, which includes an attribute for each of the layers' online instances, named with the instance's short name.

The PHP App Server layer has two instances, `php-app1` and `php-app2`. The HAProxy layer has one instance, `lb1`.

> **Note**
>
> The `instances` element contains only those instances that are in the online state when the particular stack and deployment attributes are created.

- Instance attributes – Each instance attribute contains a set of attributes that characterize the instance, such as the instance's private IP address and private DNS name. For brevity, the example shows only the `php-app2` attribute in detail; the others contain similar information.
- `applications` – A list of deployed apps, not used in this example.
- `stack` – The stack name; `MyStack` in this example.
- `instance` – The instance that these attributes are installed on; `php-app1` in this example. Recipes can use this attribute to obtain information about the instance that they are running on, such as the instance's public IP address.
- `activity` – The activity that produced the attributes; a Configure event in this example.
- `rails_stack` – The Rails stack for stacks that include a Rails App Server layer.
- `deployment` – Whether these attributes are associated with a deployment. It is set to `null` for this example because they are associated with a Configure event.
- `valid_client_activities` – A list of valid client activities.

The `opsworks` attribute is followed by several other top-level attributes, including the following:

- `opsworks_custom_cookbooks` – Whether custom cookbooks are enabled. If so, the attribute includes a list of custom recipes.
- `recipes` – The recipes that were run by this activity.
- `opsworks_rubygems` – The instance's RubyGems version.
- `ssh_users` – A list of SSH users; none in this example.
- `opsworks_bundler` – The bundler version and whether it is enabled.
- `deploy` – Information about deployment activities; none in this example.

Deployment Attributes

The attributes for a Deploy event or Execute Recipes stack command consist of the built-in stack configuration and deployment attributes, and any custom stack or deployment attributes (none for this example). The following JSON object shows the attributes from **php-app1** that are associated with a Deploy event that deployed the SimplePHP app to the stack's PHP instances. Much of the object consists of stack configuration attributes that are similar to the ones for the Configure event described in the previous section, so the example focuses primarily on the deployment-specific attributes. For a detailed description of the various attributes, see Stack Configuration and Deployment Attributes: Linux and Built-in Cookbook Attributes.

```
{
   ...
  "opsworks": {
    ...
    "activity": "deploy",
    "applications": [
      {
        "slug_name": "simplephp",
        "name": "SimplePHP",
        "application_type": "php"
      }
    ],
    "deployment": "5e6242d7-8111-40ee-bddb-00de064ab18f",
    ...
  },
  ...
{
  "ssh_users": {
  },
  "deploy": {
    "simplephpapp": {
      "application": "simplephpapp",
      "application_type": "php",
      "environment_variables": {
        "USER_ID": "168424",
        "USER_KEY": "somepassword"
      },
      "auto_bundle_on_deploy": true,
      "deploy_to": "/srv/www/simplephpapp",
      "deploying_user": "arn:aws:iam::123456789012:user/guysm",
```

```
      "document_root": null,
      "domains": [
        "simplephpapp"
      ],
      "migrate": false,
      "mounted_at": null,
      "rails_env": null,
      "restart_command": "echo 'restarting app'",
      "sleep_before_restart": 0,
      "ssl_support": false,
      "ssl_certificate": null,
      "ssl_certificate_key": null,
      "ssl_certificate_ca": null,
      "scm": {
        "scm_type": "git",
        "repository": "git://github.com/amazonwebservices/opsworks-demo-php-simple-
app.git",
        "revision": "version1",
        "ssh_key": null,
        "user": null,
        "password": null
      },
      "symlink_before_migrate": {
        "config/opsworks.php": "opsworks.php"
      },
      "symlinks": {
      },
      "database": {
      },
      "memcached": {
        "host": null,
        "port": 11211
      },
      "stack": {
        "needs_reload": false
      }
    }
  },
}
```

The `opsworks` attribute is largely identical to the example in the previous section. The following sections are most relevant to deployment:

- `activity` – The event that is associated with these attributes; a Deploy event in this example.
- `applications` – Contains a set of attributes for each app that provide the apps' names, slug names, and types.

  The slug name is the app's short name, which OpsWorks Stacks generates from the app name. The slug name for SimplePHP is simplephp.
- `deployment` – The deployment ID, which uniquely identifies a deployment.

The `deploy` attribute includes information about the apps that are being deployed. For example, the built-in Deploy recipes use the data in the `deploy` attribute to install files in the appropriate directories and create database connection files. The `deploy` attribute includes one attribute for each deployed app, named with the app's short name. Each app attribute includes the following attributes:

- `environment_variables` – Contains any environment variables that you have defined for the app. For more information, see Environment Variables.
- `domains` – By default, the domain is the app's short name, which is simplephpapp for this example. If you have assigned custom domains, they appear here as well. For more information, see Using Custom Domains.
- `application` – The app's short name.
- `scm` – This element contains the information required to download the app's files from its repository; a Git repository in this example.
- `database` – Database information, if the stack includes a database layer.
- `document_root` – The document root, which is set to `null` in this example, indicating that the root is public.
- `ssl_certificate_ca`, `ssl_support`, `ssl_certificate_key` – Indicates whether the app has SSL support. If so, the `ssl_certificate_key` and `ssl_certificate_ca` attributes are set to the corresponding certificates.
- `deploy_to` – The app's root directory.

Cookbooks 101

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

A production-level OpsWorks Stacks stack typically requires some customization, which often means implementing a custom Chef cookbook with one or more recipes, attribute files, or template files. This topic is a tutorial introduction to implementing cookbooks for OpsWorks Stacks.

For more information on how OpsWorks Stacks uses cookbooks, which includes a brief general introduction to cookbooks, see Cookbooks and Recipes. For additional information on how to implement and test Chef recipes, see Test-Driven Infrastructure with Chef, 2nd Edition.

The tutorial examples are divided into two sections:

- Cookbook Basics is a set of example walkthroughs that are intended for users who are not familiar with Chef; experienced Chef users can skip this section.

  The examples walk you through the basics of how to implement cookbooks to perform common tasks, such as installing packages or creating directories. To simplify the process, you will use a pair of useful tools, Vagrant and Test Kitchen, to run most of the examples locally in a virtual machine. Before starting Cookbook Basics, you should first read Vagrant and Test Kitchen to learn how to install and use these tools. Because Test Kitchen does not yet support Windows, the examples are all for Linux, with notes indicating how to adapt them for Windows.
- Implementing Cookbooks for OpsWorks Stacks describes how to implement recipes for OpsWorks Stacks, including for Windows stacks.

  It also includes some more advanced s such as how to use Berkshelf to manage external cookbooks. The examples are written for new Chef users, much like the examples in Cookbook Basics. However OpsWorks Stacks works a bit differently than Chef server, so we recommend that experienced Chef users at least read through this section.

Vagrant and Test Kitchen

If you are working with recipes for Linux instances, Vagrant and Test Kitchen are very useful tools for learning and initial development and testing. This provides brief descriptions of Vagrant and Test Kitchen, and points you to installation instructions and walkthroughs that will get you set up and familiarize you with the basics of how to use the tools. Although Vagrant does support Windows, Test Kitchen does not, so only Linux examples are provided for these tools.

Vagrant

Vagrant provides a consistent environment for executing and testing code on a virtual machine. It supports a wide variety of environments—called Vagrant boxes—each of which represents a configured operating system. For OpsWorks Stacks, the environments of interest are based on Ubuntu, Amazon, or Red Hat Enterprise Linux (RHEL) distributions, so the examples primarily use a Vagrant box named `opscode-ubuntu-12.04`.

Vagrant is available for Linux, Windows, and Macintosh systems, so you can use your preferred workstation to implement and test recipes on any supported operating system. The examples for this chapter were created on an Ubuntu Linux system, but translating the procedures to Windows or Macintosh systems is straightforward.

Vagrant is basically a wrapper for a virtualization provider. Most of the examples use the VirtualBox provider. VirtualBox is free and available for Linux, Windows, and Macintosh systems. The Vagrant walkthrough provides installation instructions if you do not already have VirtualBox on your system. Note that you can run Ubuntu-based environments on VirtualBox, but Amazon Linux is available only for Amazon EC2 instances. However, you can run a similar operating system such as CentOS on VirtualBox, which is useful for initial development and testing.

For information on other providers, see the Vagrant documentation. In particular, the `vagrant-aws` plug-in provider allows you to use Vagrant with Amazon EC2 instances. This provider is particularly useful for testing recipes on Amazon Linux, which is available only on Amazon EC2 instances. The `vagrant-aws` provider is free, but you must have an AWS account and pay for any AWS resources that you use.

At this point, you should go through Vagrant's Getting Started walkthrough, which describes how to install Vagrant on your workstation and teaches you the basics of how to use Vagrant. Note that the examples in this chapter do not use a Git repository, so you can omit that part of the walkthrough if you prefer.

Test Kitchen

Test Kitchen simplifies the process of executing and testing your cookbooks on Vagrant. As a practical matter, you rarely if ever need to use Vagrant directly. Test Kitchen performs most common tasks, including:

- Launching an instance in Vagrant.
- Transferring cookbooks to the instance.
- Running the cookbook's recipes on the instance.
- Testing a cookbook's recipes on the instance.
- Using SSH to log in to the instance.

Instead of installing the Test Kitchen gem directly, we recommend installing Chef DK. In addition to Chef itself, this package includes Test Kitchen, Berkshelf, ChefSpec, and several other useful tools.

At this point, you should go through Test Kitchen's Getting Started walkthrough, which teaches you the basics of how to use Test Kitchen to execute and test recipes.

> **Note**
>
> The examples in this chapter use Test Kitchen as a convenient way to run recipes. If you prefer, you can stop the Getting Started walkthrough after completing the Manually Verifying section, which covers everything you need to know for the examples. However, Test Kitchen is primarily a testing platform that supports test frameworks such as bash automated test system (BATS). You should complete the remainder of the walkthrough at some point to learn how to use Test Kitchen to test your recipes.

Cookbook Basics

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can use cookbooks to accomplish a wide variety of tasks. The following topics assume that you are new to Chef, and describe how to use cookbooks to accomplish some common tasks. Because Test Kitchen does not yet support Windows, the examples are all for Linux, with notes indicating how to adapt them for Windows. If you are new to Chef, we recommend going through these examples, even if you will be working with Windows. Most of the examples in this topic can be used on Windows instances with some modest changes, which are noted in the examples. All of the examples run in a virtual machine, so you don't even need to have a Linux computer. Just install Vagrant and Test Kitchen on your regular workstation.

> **Note**
>
> If you want to run these recipes on a Windows instance, the simplest approach is to create a Windows stack and run the recipes on one of the stack's instances. For more information on how to run recipes on an OpsWorks Stacks Windows instance, see Running a Recipe on a Windows Instance.

Before continuing, make sure that you have installed Vagrant and Test Kitchen, and gone through their Getting Started walkthroughs. For more information, see Vagrant and Test Kitchen.

Topics

- Recipe Structure
- Example 1: Installing Packages
- Example 2: Managing Users
- Example 3: Creating Directories
- Example 4: Adding Flow Control
- Example 5: Using Attributes
- Example 6: Creating Files
- Example 7: Running Commands and Scripts
- Example 8: Managing Services
- Example 9: Using Amazon EC2 Instances
- Next Steps

Recipe Structure

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

A cookbook is primarily a set of *recipes*, which can perform a wide variety of tasks on an instance. To clarify how to implement recipes, it's useful to look at a simple example. The following is the setup recipe for the built-in HAProxy layer. Just focus on the overall structure at this point and don't worry too much about the details; they will be covered in the subsequent examples.

```
package 'haproxy' do
  action :install
end

if platform?('debian','ubuntu')
  template '/etc/default/haproxy' do
    source 'haproxy-default.erb'
    owner 'root'
    group 'root'
    mode 0644
  end
end

include_recipe 'haproxy::service'

service 'haproxy' do
  action [:enable, :start]
end

template '/etc/haproxy/haproxy.cfg' do
  source 'haproxy.cfg.erb'
  owner 'root'
  group 'root'
  mode 0644
```

```
  notifies :restart, "service[haproxy]"
end
```

> **Note**
>
> For this and other examples of working recipes and related files, see the OpsWorks Stacks built-in recipes.

The example highlights the key recipe elements, which are described in the following sections.

Topics

- Resources
- Flow Control
- Included Recipes

Resources

Recipes consist largely of a set of Chef *resources*. Each one specifies a particular aspect of the instance's final state, such as a package to be installed or a service to be started. The example has four resources:

- A `package` resource, which represents an installed package, an HAProxy server for this example.
- A `service` resource, which represents a service, the HAProxy service for this example.
- Two `template` resources, which represent files that are to be created from a specified template, two HAProxy configuration files for this example.

Resources provide a declarative way to specify the instance state. Behind the scenes, each resource has an associated *provider* that performs the required tasks, such as installing packages, creating and configuring directories, starting services, and so on. If the details of the task depend on the particular operating system, the resource has multiple providers and uses the appropriate one for the system. For example, on a Red Hat Linux system the `package` provider uses `yum` to install packages. On a Ubuntu Linux system, the `package` provider uses `apt-get`.

You implement a resource as a Ruby code block with the following general format.

```
resource_type "resource_name" do
  attribute1 'value1'
  attribute2 'value2'
  ...
  action :action_name
  notifies : action 'resource'
end
```

The elements are:

Resource type

(Required) The example includes three resource types, `package`, `service`, and `template`.

Resource name

(Required) The name identifies the particular resource and is sometimes used as a default value for one of the attributes. In the example, `package` represents a package resource named `haproxy` and the first `template` resource represents a configuration file named `/etc/default/haproxy`.

Attributes

(Optional) Attributes specify the resource configuration and vary depending on the resource type and how you want to configure the resource.

- The example's `template` resources explicitly define a set of attributes that specify the created file's source, owner, group, and mode.
- The example's `package` and `service` resources do not explicitly define any attributes.

  The resource name is typically the default value for a required attribute and is sometimes all that is needed. For example, the resource name is the default value for the `package` resource's `package_name` attribute, which is the only required attribute.

There are also some specialized attributes called guard attributes, which specify when the resource provider is to take action. For example, the `only_if` attribute directs the resource provider to take action only if a specified condition is met. The HAProxy recipe does not use guard attributes, but they are used by several of the following examples.

Actions and Notifications

(Optional) Actions and notifications specify what tasks the provider is to perform.

- `action` directs the provider to take a specified action, such as install or create.

Each resource has a set of actions that depend on the particular resource, one of which is the default action. In the example, the `package` resource's action is `install`, which directs the provider to install the package. The first `template` resource has no `action` element, so the provider takes the default `create` action.

- `notifies` directs another resource's provider to perform an action, but only if the resource's state has changed.

  `notifies` is typically used with resources such as `template` and `file` to perform tasks such as restarting a service after modifying a configuration file. Resources do not have default notifications. If you want a notification, the resource must have an explicit `notifies` element. In the HAProxy recipe, the second `template` resource notifies the haproxy `service` resource to restart the HAProxy service if the associated configuration file has changed.

Resources sometimes depend on operating system.

- Some resources can be used only on Linux or Windows systems.

  For example, package installs packages on Linux systems and windows_package installs packages on Windows systems.
- Some resources can be used with any operating system, but have attributes that are specific to a particular system.

  For example, the file resource can be used on either Linux or Windows systems, but has separate sets of attributes for configuring permissions.

For descriptions of the standard resources, including the available attributes, actions, and notifications for each resource, see About Resources and Providers.

Flow Control

Because recipes are Ruby applications, you can use Ruby control structures to incorporate flow control into a recipe. For example, you can use Ruby conditional logic to have the recipe behave differently on different systems. The HAProxy recipe includes an `if` block that uses a `template` resource to create a configuration file, but only if the recipe is running on a Debian or Ubuntu system.

Another common scenario is using a loop to execute a resource multiple times with different attribute settings. For example, you can create a set of directories by using a loop to execute a `directory` resource multiple times with different directory names.

> **Note**
>
> If you aren't familiar with Ruby, see Just Enough Ruby for Chef, which covers what you need to know for most recipes.

Included Recipes

`include_recipe` includes other recipes in your code, which allows you to modularize your recipes and reuse the same code in multiple recipes. When you run the host recipe, Chef replaces each `include_recipe` element with the specified recipe's code before it executes the host recipe. You identify an included recipe by using the standard Chef `cookbook_name::recipe_name` syntax, where `recipe_name` omits the `.rb` extension. The example includes one recipe, `haproxy::service`, which represents the HAProxy service.

> **Note**
>
> If you use `include_recipe` in recipes running on Chef 11.10 and later to include a recipe from another cookbook, you must use a `depends` statement to declare the dependency in the cookbook's `metadata.rb` file. For more information, see Implementing Recipes: Chef 11.10.

Example 1: Installing Packages

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Package installation is one of the more common uses of recipes and can be quite simple, depending on the package. For example, the following recipe installs Git on a Linux system.

```
package 'git' do
  action :install
end
```

The package resource handles package installation. For this example, you don't need to specify any attributes. The resource name is the default value for the `package_name` attribute, which identifies the package. The `install` action directs the provider to install the package. You could make the code even simpler by skipping `install`; it's the `package` resource's default action. When you run the recipe, Chef uses the appropriate provider to install the package. On the Ubuntu system that you will use for the example, the provider installs Git by calling `apt-get`.

> **Note**
>
> Installing software on a Windows system requires a somewhat different procedure. For more information, see Installing Windows Software.

To use Test Kitchen to run this recipe in Vagrant, you first need to set up a cookbook and initialize and configure Test Kitchen. The following is for a Linux system, but the procedure is essentially similar for Windows and Macintosh systems. Start by opening a Terminal window; all of the examples in this chapter use command-line tools.

To prepare the cookbook

1. In your home directory, create a subdirectory named `opsworks_cookbooks`, which will contain all the cookbooks for this chapter. Then create a subdirectory for this cookbook named `installpkg` and navigate to it.
2. In `installpkg`, create a file named `metadata.rb` that contains the following code.

   ```
   name "installpkg"
   version "0.1.0"
   ```

For simplicity, the examples in this chapter just specify the cookbook name and version, but `metadata.rb` can contain a variety of cookbook metadata. For more information, see About Cookbook Metadata.

> **Note**
>
> Make sure to create `metadata.rb` before you initialize Test Kitchen; it uses the data to create the default configuration file.

3. In `installpkg`, run `kitchen init`, which initializes Test Kitchen and installs the default Vagrant driver.
4. The `kitchen init` command creates a YAML configuration file in `installpkg` named `.kitchen.yml`. Open the file in your favorite text editor. The `.kitchen.yml` file includes a `platforms` section that specifies which systems to run the recipes on. Test Kitchen creates an instance and runs the specified recipes on each platform.

> **Note**
>
> By default, Test Kitchen runs recipes one platform at a time. If you add a `-p` argument to any command that creates an instance, Test Kitchen will run the recipes on every platform, in parallel.

A single platform is sufficient for this example, so edit `.kitchen.yml` to remove the `centos-6.4` platform. Your `.kitchen.yml` file should now look like this:

```
---
driver:
  name: vagrant

provisioner:
  name: chef_solo

platforms:
  - name: ubuntu-12.04

suites:
```

```
  - name: default
    run_list:
      - recipe[installpkg::default]
    attributes:
```

Test Kitchen runs only those recipes that are in the `.kitchen.yml` run list. You identify recipes by using the [*cookbook_name*::*recipe_name*] format, where *recipe_name* omits the `.rb` extension. Initially, the `.kitchen.yml` run list contains the cookbook's default recipe, `installpkg::default`. That's the recipe that you are going to implement, so you don't need to modify the run list.

5. Create a subdirectory of `installpkg` named `recipes`.

   If a cookbook contains recipes—most do—they must be in the `recipes` subdirectory.

You can now add the recipe to the cookbook and use Test Kitchen to run it on an instance.

To run the recipe

1. Create a file named `default.rb` that contains the Git installation example code from the beginning of the section and save it to the `recipes` subdirectory.
2. In the `installpkg` directory, run `kitchen converge`. This command starts a new Ubuntu instance in Vagrant, copies your cookbooks to the instance, and initiates a Chef run to execute the recipes in the `.kitchen.yml` run list.
3. To verify that the recipe was successful, run `kitchen login`, which opens an SSH connection to the instance. Then run `git --version` to verify that Git was successfully installed. To return to your workstation, run `exit`.
4. When you are finished, run `kitchen destroy` to shut down the instance. The next example uses a different cookbook.

This example was a good way to get started, but it is especially simple. Other packages can be more complicated to install; you might need to do any or all of the following:

- Create and configure a user.
- Create one or more directories for data, logs, and so on.
- Install one or more configuration files.
- Specify a different package name or attribute values for different operating systems.

- Start a service and then restart it as needed.

The following examples describe how to address these issues, along with some other useful operations.

Example 2: Managing Users

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Another simple task is managing users on an instance. The following recipe adds a new user to a Linux instance.

```
user "myuser" do
  home "/home/newuser"
  shell "/bin/bash"
end
```

You use a user resource to manage users on both Linux and Windows systems, although some attributes apply to only one system. The example creates a user named `myuser` and specifies their home directory and shell. There is no action specified, so the resource uses the default `create` action. You can add attributes to `user` to specify a variety of other settings, such as their password or group ID. You can also use `user` for related user-management tasks such as modifying user settings or deleting users. For more information, see user.

To run the recipe

1. Create a directory within `opsworks_cookbooks` named `newuser` and navigate to it.
2. Create a `metadata.rb` file that contains the following code and save it to `newuser`.

   ```
   name "newuser"
   ```

```
version "0.1.0"
```

3. Initialize and configure Test Kitchen, as described in Example 1: Installing Packages, and add a `recipes` directory inside the `newuser` directory.
4. Add `default.rb` file with the example recipe to the cookbook's `recipes` directory .
5. Run `kitchen converge` to execute the recipe.
6. Use `kitchen login` to log in to the instance and verify the new user's existence by running `cat /etc/passwd`. The `myuser` user should be at the bottom of the file.

Example 3: Creating Directories

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

When you install a package on an instance, you often need to create some configuration files and place them in the appropriate directories. However, those directories might not exist yet. You might also need to create directories for data, log files, and so on. For example, you first boot the Ubuntu system that you use for most of the examples, the `/srv` directory has no subdirectories. If you are installing an application server, you will probably want a `/srv/www/` directory and perhaps some subdirectories for data files, logs, and so on. The following recipe creates `/srv/www/` on an instance.

```
directory "/srv/www/" do
  mode 0755
  owner 'root'
  group 'root'
  action :create
end
```

You use a `directory` resource to create and configure directories on both Linux and Windows systems, although some attributes are used differently. The resource name is the default value for

the resource's `path` attribute, so the example creates `/srv/www/` and specifies its `mode`, `owner`, and `group` properties.

To run the recipe

1. Create a directory inside `opsworks_cookbooks` named `createdir` and navigate to it.
2. Initialize and configure Test Kitchen, as described in Example 1: Installing Packages, and add a `recipes` directory within `createdir`.
3. Add a `default.rb` file with the recipe code to the cookbook's `recipes` subdirectory.
4. Run `kitchen converge` to execute the recipe.
5. Run `kitchen login`, navigate to `/srv` and verify that it has a `www` subdirectory.
6. Run `exit` to return to your workstation but leave the instance running.

Note

To create a directory relative to your home directory on the instance, use `#{ENV['HOME']}` to represent the home directory. For example, the following creates the `~/shared` directory.

```
directory "#{ENV['HOME']}/shared" do
  ...
end
```

Suppose that you want to create a more deeply nested directory, such as `/srv/www/shared`. You could modify the preceding recipe as follows.

```
directory "/srv/www/shared" do
  mode 0755
  owner 'root'
  group 'root'
  action :create
end
```

To run the recipe

1. Replace the code in `default.rb` with the preceding recipe.
2. Run `kitchen converge` from the `createdir` directory.
3. To verify that the directory was indeed created, run `kitchen login`, navigate to `/srv/www`, and verify that it contains a `shared` subdirectory.
4. Run `kitchen destroy` to shut the instance down.

You will notice the `kitchen converge` command ran much faster. That's because the instance is already running, so there's no need to boot the instance, install Chef, and so on. Test Kitchen just to copies the updated cookbook to the instance and starts a Chef run.

Now run `kitchen converge` again, which executes the recipe on a fresh instance. You'll now see the following result.

```
Chef Client failed. 0 resources updated in 1.908125788 seconds
[2014-06-20T20:54:26+00:00] ERROR: directory[/srv/www/shared] (createdir::default line
 1) had an error: Chef::Exceptions::EnclosingDirectoryDoesNotExist: Parent directory /
srv/www does not exist, cannot create /srv/www/shared
[2014-06-20T20:54:26+00:00] FATAL: Chef::Exceptions::ChildConvergeError: Chef run
 process exited unsuccessfully (exit code 1)
>>>>>> Converge failed on instance <default-ubuntu-1204>.
>>>>>> Please see .kitchen/logs/default-ubuntu-1204.log for more details
>>>>>> ------Exception-------
>>>>>> Class: Kitchen::ActionFailed
>>>>>> Message: SSH exited (1) for command: [sudo -E chef-solo --config /tmp/kitchen/
solo.rb --json-attributes /tmp/kitchen/dna.json  --log_level info]
>>>>>> ----------------------
```

What happened? The problem is that by default, a `directory` resource can create only one directory at a time; it can't create a chain of directories. The reason the recipe worked earlier is that the very first recipe you ran on the instance had already created `/srv/www`, so creating `/srv/www/shared` created only one subdirectory.

> **Note**
>
> When you run `kitchen converge`, make sure you know whether you are running your recipes on a new or existing instance. You might get different results.

To create a chain of subdirectories, add a `recursive` attribute to `directory` and set it to `true`. The following recipe creates `/srv/www/shared` directly on a clean instance.

```
directory "/srv/www/shared" do
  mode 0755
  owner 'root'
  group 'root'
  recursive true
  action :create
end
```

Example 4: Adding Flow Control

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Some recipes are just a series of Chef resources. In that case, when you run the recipe, it simply executes each of the resource providers in sequence. However, it's often useful to have a more sophisticated execution path. The following are two common scenarios:

- You want a recipe to execute the same resource multiple times with different attribute settings.
- You want to use different attribute settings on different operating systems.

You can address scenarios such as these by incorporating Ruby control structures into the recipe. This section shows how to modify the recipe from Example 3: Creating Directories to address both scenarios.

Topics

- Iteration
- Conditional Logic

Iteration

Example 3: Creating Directories showed how to use a `directory` resource to create a directory or chain of directories. However, suppose that you want to create two separate directories, `/srv/www/config` and `/srv/www/shared`. You could implement a separate directory resource for each directory, but that approach can get cumbersome if you want to create very many directories. The following recipe shows a simpler way to handle the task.

```
[ "/srv/www/config", "/srv/www/shared" ].each do |path|
  directory path do
    mode 0755
    owner 'root'
    group 'root'
    recursive true
    action :create
  end
end
```

Instead of using a separate directory resource for each subdirectory, the recipe uses a string collection that contains the subdirectory paths. The Ruby `each` method executes the resource once for each collection element, starting with the first one. The element's value is represented in the resource by the `path` variable, which in this case represents the directory path. You can easily adapt this example to create any number of subdirectories.

To run the recipe

1. Stay in `createdir` directory; you'll be using that cookbook for the next several examples.
2. If you haven't done so already, run `kitchen destroy` so you are starting with a clean instance.

3. Replace the code in `default.rb` with the example and run `kitchen converge`.
4. Log in to the instance; you will see the newly created directories under `/srv`.

You can use a hash table to specify two values for each iteration. The following recipe creates `/srv/www/config` and `/srv/www/shared`, each with a different mode.

```
{ "/srv/www/config" => 0644, "/srv/www/shared" => 0755 }.each do |path, mode_value|
  directory path do
    mode mode_value
    owner 'root'
    group 'root'
    recursive true
    action :create
  end
end
```

To run the recipe

1. If you haven't done so already, run `kitchen destroy` so you are starting with a clean instance.
2. Replace the code in `default.rb` with the example and run `kitchen converge`.
3. Log in to the instance; you will see the newly created directories under `/srv` with the specified modes.

> **Note**
>
> OpsWorks Stacks recipes commonly use this approach to extract values from the stack configuration and deployment JSON—which is basically a large hash table—and insert them in a resource. For an example, see Deploy Recipes.

Conditional Logic

You can also use Ruby conditional logic to create multiple execution branches. The following recipe uses `if-elsif-else` logic to extend the previous example so that it creates a subdirectory named `/srv/www/shared`, but only on Debian and Ubuntu systems. For all other systems, it logs an error message that is displayed in the Test Kitchen output.

```
if platform?("debian", "ubuntu")
  directory "/srv/www/shared" do
    mode 0755
    owner 'root'
    group 'root'
    recursive true
    action :create
  end
else
  log "Unsupported system"
end
```

To run the example recipe

1. If your instance is still up, run `kitchen destroy` to shut it down.
2. Replace the code in `default.rb` with the example code.
3. Edit `.kitchen.yml` to add a CentOS 6.4 system to the platform list. The file's `platforms` section should now look like.

   ```
   ...
   platforms:
     - name: ubuntu-12.04
     - name: centos-6.4
   ...
   ```

4. Run `kitchen converge`, which will create an instance and run the recipes for each platform in `.kitchen.yml`, in sequence.

   > **Note**
   >
   > If you want to converge just one instance, add the instance name as a parameter. For example, to converge the recipe only on the Ubuntu platform, run `kitchen converge default-ubuntu-1204`. If you forget the platform names, just run `kitchen list`.

You should see your log message in the CentOS part of the Test Kitchen output, which will look something like the following:

```
...
Converging 1 resources
Recipe: createdir::default
* log[Unsupported system] action write[2014-06-23T19:10:30+00:00] INFO: Processing
 log[Unsupported system] action write (createdir::default line 12)
[2014-06-23T19:10:30+00:00] INFO: Unsupported system

[2014-06-23T19:10:30+00:00] INFO: Chef Run complete in 0.004972162 seconds
```

You can now log in to the instances and verify that the directories were or were not created. However, you can't simply run `kitchen login` now. You must specify which instance by appending the platform name, for example, `kitchen login default-ubuntu-1204`.

> **Note**
>
> If a Test Kitchen command takes an instance name, you don't need to type the complete name. Test Kitchen treats an instance name as a Ruby regular expression, so you just need enough characters to provide a unique match. For example, you can converge just the Ubuntu instance by running `kitchen converge ub` or log in to the CentOS instance by running `kitchen login 64`.

The question you probably have at this point is how the recipe knows which platform it is running on. Chef runs a tool called Ohai for every run that collects system data, including the platform, and represents it as a set of attributes in a structure called the *node object*. The Chef `platform?` method compares the systems in parentheses against the Ohai platform value, and returns true if one of them matches.

You can reference the value of a node attribute directly in your code by using `node['attribute_name']`. The platform value, for example, is represented by `node['platform']`. You could, for example, have written the preceding example as follows.

```
if node[:platform] == 'debian' or node[:platform] == 'ubuntu'
  directory "/srv/www/shared" do
```

```
    mode 0755
    owner 'root'
    group 'root'
    recursive true
    action :create
  end
else
  log "Unsupported system"
end
```

A common reason for including conditional logic in a recipe is to accommodate the fact that different Linux families sometimes use different names for packages, directories, and so on. For example, the Apache package name is `httpd` on CentOS systems and `apache2` on Ubuntu systems.

If you just need a different string for different systems, the Chef `value_for_platform` method is a simpler solution than `if-elsif-else`. The following recipe creates a `/srv/www/shared` directory on CentOS systems, a `/srv/www/data` directory on Ubuntu systems, and `/srv/www/config` on all others.

```
data_dir = value_for_platform(
  "centos" => { "default" => "/srv/www/shared" },
  "ubuntu" => { "default" => "/srv/www/data" },
  "default" => "/srv/www/config"
)
directory data_dir do
  mode 0755
  owner 'root'
  group 'root'
  recursive true
  action :create
end
```

`value_for_platform` assigns the appropriate path to `data_dir` and the `directory` resource uses that value to create the directory.

To run the example recipe

1. If your instance is still up, run `kitchen destroy` to shut it down.
2. Replace the code in `default.rb` with the example code.

3. Run `kitchen converge` and then login to each instance to verify that the appropriate directories are present.

Example 5: Using Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The recipes in the preceding sections used hard-coded values for everything other than the platform. This approach can be inconvenient if, for example, you want to use the same value in more than one recipe. You can define values separately from recipes by including an attribute file in your cookbook.

An attribute file is a Ruby application that assigns values to one or more attributes. It must be in the cookbook's `attributes` folder. Chef incorporates the attributes into the node object and any recipe can use the attribute values by referencing the attribute. This topic shows how to modify the recipe from Iteration to use attributes. Here's the original recipe for reference.

```
[ "/srv/www/config", "/srv/www/shared" ].each do |path|
  directory path do
    mode 0755
    owner 'root'
    group 'root'
    recursive true
    action :create
  end
end
```

The following defines attributes for the subdirectory name, mode, owner, and group values.

```
default['createdir']['shared_dir'] = 'shared'
```

```
default['createdir']['config_dir'] = 'config'
default['createdir']['mode'] = 0755
default['createdir']['owner'] = 'root'
default['createdir']['group'] = 'root'
```

Note the following:

- Each definition starts with an *attribute type*.

  If an attribute is defined more than once—perhaps in different attribute files—the attribute type specifies the attribute's precedence, which determines which definition is incorporated into the node object. For more information, see Attribute Precedence. All the definitions in this example have the `default` attribute type, which is the usual type for this purpose.
- The attributes have nested names.

  The node object is basically a hash table that can be nested arbitrarily deeply, so attribute names can be and commonly are nested. This attribute file follows a standard practice of using a nested name with the cookbook name, `createdir`, as the first element.

The reason for using createdir as the attribute's first element is that when you do a Chef run, Chef incorporates the attributes from every cookbook into the node object. With OpsWorks Stacks, the node object includes a large number of attributes from the built-in cookbooks in addition to any attributes that you define. Including the cookbook name in the attribute name reduces the risk of a name collision with attributes from another cookbook, especially if your attribute has a name like `port` or `user`. Don't name an attribute something like `[:apache2][:user]`, for example, unless you want to override that attribute's value. For more information, see Using Custom Cookbook Attributes.

The following example shows the original recipe using attributes instead of hard-coded values.

```
[ "/srv/www/#{node['createdir']['shared_dir']}", "/srv/www/#{node['createdir']
['config_dir']}" ].each do |path|
  directory path do
    mode node['createdir']['mode']
    owner node['createdir']['owner']
    group node['createdir']['group']
    recursive true
    action :create
  end
```

```
end
```

> **Note**
>
> If you want to incorporate an attribute value into a string, wrap it with `#{}`. In the preceding example, `#{node['createdir']['shared_dir']}` appends "shared" to "/srv/www/".

To run the recipe

1. Run `kitchen destroy` to start with a clean instance.
2. Replace the code in `recipes/default.rb` with the preceding recipe example.
3. Create a subdirectory of `createdir` named `attributes` and add a file named `default.rb` that contains the attribute definitions.
4. Edit `.kitchen.yml` to remove CentOS from the platforms list.
5. Run `kitchen converge` and then log in to the instance and verify that `/srv/www/shared` and `/srv/www/config` are there.

> **Note**
>
> With OpsWorks Stacks, defining values as attributes provides an additional benefit; you can use custom JSON to override those values on a per-stack or even per-deployment basis. This can be useful for a variety of purposes, including the following:
>
> - You can customize the behavior of your recipes, such as configuration settings or user names, without having to modify the cookbook.
>
> You can, for example, use the same cookbook for different stacks and use custom JSON to specify key configuration settings for a particular stack. This saves you the time and effort required to modify the cookbook or use a different cookbook for each stack.
> - You don't have to put potentially sensitive information such as database passwords in your cookbook repository.
>
> You can instead use an attribute to define a default value and then use custom JSON to override that value with the real one.

For more information on how to use custom JSON to override attributes, see Overriding Attributes.

The attribute file is named `default.rb` because it is a Ruby application, if a rather simple one. That means you can, for example, use conditional logic to specify attribute values based on the operating system. In Conditional Logic, you specified a different subdirectory name for different Linux families in the recipe. With an attribute file, you can instead put the conditional logic in the attribute file.

The following attribute file uses `value_for_platform` to specify a different `['shared_dir']` attribute value, depending on the operating system. For other conditions, you can use Ruby `if-elsif-else` logic or a `case` statement.

```
data_dir = value_for_platform(
  "centos" => { "default" => "shared" },
  "ubuntu" => { "default" => "data" },
  "default" => "user_data"
)
default['createdir']['shared_dir'] = data_dir
default['createdir']['config_dir'] = "config"
default['createdir']['mode'] = 0755
default['createdir']['owner'] = 'root'
default['createdir']['group'] = 'root'
```

To run the recipe

1. Run `kitchen destroy` to start with a fresh instance.
2. Replace the code in `attributes/default.rb` with the preceding example.
3. Edit `.kitchen.yml` to add a CentOS platform to the platforms section, as described in Conditional Logic.
4. Run `kitchen converge`, and then log in to the instances to verify that the directories are there.

When you are finished, run `kitchen destroy` to terminate the instance. The next example uses a new cookbook.

Example 6: Creating Files

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

After you have created directories, you often need to populate them with configuration files, data files, and so on. This topic shows two ways to install files on an instance.

Topics

- Installing a File from a Cookbook
- Creating a File from a Template

Installing a File from a Cookbook

The simplest way to install a file on an instance is to use a `cookbook_file` resource, which copies a file from the cookbook to a specified location on the instance for both Linux and Windows systems. This example extends the recipe from Example 3: Creating Directories to add a data file to `/srv/www/shared` after the directory is created. For reference, here is the original recipe.

```
directory "/srv/www/shared" do
  mode 0755
  owner 'root'
  group 'root'
  recursive true
  action :create
end
```

To set up the cookbook

1. Inside the `opsworks_cookbooks` directory, create a directory named `createfile` and navigate to it.
2. Add a `metadata.rb` file to `createfile` with the following content.

```
name "createfile"
version "0.1.0"
```

3. Initialize and configure Test Kitchen, as described in Example 1: Installing Packages, and remove CentOS from the `platforms` list.
4. Add a `recipes` subdirectory to `createfile`.

The file to be installed contains the following JSON data.

```
{
  "my_name" : "myname",
  "your_name" : "yourname",
  "a_number" : 42,
  "a_boolean" : true
}
```

To set up the data file

1. Add a `files` subdirectory to `createfile` and a `default` subdirectory to `files`. Any file that you install with `cookbook_file` must be in a subdirectory of `files`, such as `files/default` in this example.

   > **Note**
   >
   > If you want to specify different files for different systems, you can put each system-specific file in a subfolder named for the system, such as `files/ubuntu`. The `cookbook_file` resource copies the appropriate system-specific file, if it exists, and otherwise uses the `default` file. For more information, see cookbook_file.

2. Create a file named `example_data.json` with the JSON from the preceding example and add it to `files/default`.

The following recipe copies `example_data.json` to a specified location.

```
directory "/srv/www/shared" do
```

```
  mode 0755
  owner 'root'
  group 'root'
  recursive true
  action :create
end

cookbook_file "/srv/www/shared/example_data.json" do
  source "example_data.json"
  mode 0644
  action :create_if_missing
end
```

After the directory resource creates `/srv/www/shared`, the `cookbook_file` resource copies `example_data.json` to that directory and also sets the file's user, group, and mode.

> **Note**
>
> The `cookbook_file` resource introduces a new action: `create_if_missing`. You could also use a `create` action, but that overwrites an existing file. If you don't want to overwrite anything, use `create_if_missing`, which installs `example_data.json` only if it does not already exist.

To run the recipe

1. Run `kitchen destroy` to start with a fresh instance.
2. Create a `default.rb` file that contains the preceding recipe and save it to `recipes`.
3. Run `kitchen converge`, then log in to the instance to verify that `/srv/www/shared` containsexample_data.json.

Creating a File from a Template

The `cookbook_file` resource is useful for some purposes, but it just installs whatever file you have in the cookbook. A `template` resource provides a more flexible way to install a file on a Windows or Linux instance by creating it dynamically from a template. You can then determine the details of the file's contents at runtime and change them as needed. For example, you might want a configuration file to have a particular setting when you start the instance and modify the setting later when you add more instances to the stack.

This example modifies the `createfile` cookbook to use a `template` resource to install a slightly modified version of `example_data.json`.

Here's what the installed file will look like.

```
{
  "my_name" : "myname",
  "your_name" : "yourname",
  "a_number" : 42,
  "a_boolean" : true,
  "a_string" : "some string",
  "platform" : "ubuntu"
}
```

Template resources are typically used in conjunction with attribute files, so the example uses one to define the following values.

```
default['createfile']['my_name'] = 'myname'
default['createfile']['your_name'] = 'yourname'
default['createfile']['install_file'] = true
```

To set up the cookbook

1. Delete the `createfile` cookbook's `files` directory and its contents.
2. Add an `attributes` subdirectory to `createfile` and add a `default.rb` file to `attributes` that contains the preceding attribute definitions.

A template is a `.erb` file that is basically a copy of the final file, with some of the contents represented by placeholders. When the `template` resource creates the file, it copies the template's contents to the specified file, and overwrites the placeholders with their assigned values. Here's the template for `example_data.json`.

```
{
  "my_name" : "<%= node['createfile']['my_name'] %>",
  "your_name" : "<%= node['createfile']['your_name'] %>",
  "a_number" : 42,
```

```
  "a_boolean" : <%= @a_boolean_var %>,
  "a_string" : "<%= @a_string_var %>",
  "platform" : "<%= node['platform'] %>"
}
```

The <%=...%> values are the placeholders.

- <%=node[...]%> represents a node attribute value.

  For this example, the "your_name" value is a placeholder that represents one of the attribute values from the cookbook's attribute file.
- <%=@...%> represents the value of a variable that is defined in the template resource, as discussed shortly.

To create the template file

1. Add a templates subdirectory to the createfile cookbook and a default subdirectory to templates.

   > **Note**
   >
   > The templates directory works much like the files directory. You can put system-specific templates in a subdirectory such as ubuntu that is named for the system. The template resource uses the appropriate system-specific template if it exists and otherwise uses the default template.

2. Create a file named example_data.json.erb and put in the templates/default directory. The template name is arbitrary, but you usually create it by appending .erb to the file name, including any extensions.

The following recipe uses a template resource to create /srv/www/shared/example_data.json.

```
directory "/srv/www/shared" do
  mode 0755
  owner 'root'
  group 'root'
  recursive true
```

```
  action :create
end

template "/srv/www/shared/example_data.json" do
  source "example_data.json.erb"
  mode 0644
  variables(
    :a_boolean_var => true,
    :a_string_var => "some string"
  )
  only_if {node['createfile']['install_file']}
end
```

The `template` resource creates `example_data.json` from a template and installs it in `/srv/www/shared`.

- The template name, `/srv/www/shared/example_data.json`, specifies the installed file's path and name.
- The `source` attribute specifies the template used to create the file.
- The `mode` attribute specifies the installed file's mode.
- The resource defines two variables, `a_boolean_var` and `a_string_var`.

  When the resource creates `example_data.json`, it overwrites the variable placeholders in the template with the corresponding values from the resource.
- The `only_if` *guard* attribute directs the resource to create the file only if `['createfile'] ['install_file']` is set to `true`.

To run the recipe

1. Run `kitchen destroy` to start with a fresh instance.
2. Replace the code in `recipes/default.rb` with the preceding example.
3. Run `kitchen converge`, then log in to the instance to verify that the file is in `/srv/www/shared` and has the correct content.

When you are finished, run `kitchen destroy` to shut down the instance. The next section uses a new cookbook.

Example 7: Running Commands and Scripts

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Chef resources can handle a wide variety of tasks on an instance, but it is sometimes preferable to use a shell command or a script. For example, you might already have scripts that you use to accomplish certain tasks, and it will be easier to continue using them rather than implement new code. This section shows how to run commands or scripts on an instance.

Topics

- Running Commands
- Running Scripts

Running Commands

The `script` resource runs one or more commands . It supports the csh, bash, Perl, Python, and Ruby command interpreters, so it can be used on either Linux or Windows systems as long as they have the appropriate interpreters installed. This topic shows how to run a simple bash command on a Linux instance. Chef also supports powershell_script and batch resources to run scripts on Windows. For more information, see Running a Windows PowerShell Script.

To get started

1. Inside the `opsworks_cookbooks` directory, create a directory named `script` and navigate to it.
2. Add a `metadata.rb` file to `script` with the following content.

```
name "script"
version "0.1.0"
```

3. Initialize and configure Test Kitchen, as described in Example 1: Installing Packages, and remove CentOS from the `platforms` list.
4. Inside `script`, create a directory named `recipes`.

You can run commands by using the `script` resource itself, but Chef also supports a set of command interpreter-specific versions of the resource, which are named for the interpreter. The following recipe uses a `bash` resource to run a simple bash script.

```
bash "install_something" do
  user "root"
  cwd "/tmp"
  code <<-EOH
    touch somefile
  EOH
  not_if do
    File.exists?("/tmp/somefile")
  end
end
```

The `bash` resource is configured as follows.

- It uses the default action, `run`, which runs the commands in the `code` block.

  This example has one command, `touch somefile`, but a `code` block can contain multiple commands.
- The `user` attribute specifies the user that executes the command.
- The `cwd` attribute specifies the working directory.

  For this example, `touch` creates a file in the `/tmp` directory.
- The `not_if` guard attribute directs the resource to take no action if the file already exists.

To run the recipe

1. Create a `default.rb` file that contains the preceding example code and save it to `recipes`.
2. Run `kitchen converge`, then log in to the instance to verify that the file is in `/tmp`.

Running Scripts

The `script` resource is convenient, especially if you need to run only one or two commands, but it's often preferable to store the script in a file and execute the file. The `execute` resource runs a specified executable file, including script files, on Linux or Windows. This topic modifies the `script` cookbook from the preceding example to use `execute` to run a simple shell script. You can easily extend the example to more complex scripts, or other types of executable file.

To set up the script file

1. Add a `files` subdirectory to `script` and a `default` subdirectory to `files`.
2. Create a file named `touchfile` that contains the following and add it to `files/default`. A common Bash interpreter line is used in this example, but substitute an interpreter that works for your shell environment if necessary.

   ```
   #!/usr/bin/env bash
   touch somefile
   ```

   The script file can contain any number of commands. For convenience, this example script has only a single `touch` command.

The following recipe executes the script.

```
cookbook_file "/tmp/touchfile" do
  source "touchfile"
  mode 0755
end

execute "touchfile" do
  user "root"
  cwd "/tmp"
  command "./touchfile"
end
```

The `cookbook_file` resource copies the script file to `/tmp` and sets the mode to make the file executable. The `execute` resource then executes the file as follows:

- The `user` attribute specifies the command's user (`root` in this example).

- The `cwd` attribute specifies the working directory (`/tmp` in this example).
- The `command` attribute specifies the script to be executed (`touchfile` in this example), which is located in the working directory.

To run the recipe

1. Replace the code in `recipes/default.rb` with the preceding example.
2. Run `kitchen converge`, then log in to the instance to verify that `/tmp` now contains the script file, with the mode set to 0755, and `somefile`.

When you are finished, run `kitchen destroy` to shut down the instance. The next section uses a new cookbook.

Example 8: Managing Services

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Packages such as application servers typically have an associated service that must be started, stopped, restarted, and so on. For example, you need to start the Tomcat service after installing the package or after the instance finishes booting, and restart the service each time you modify the configuration file. This topic discusses the basics of how to manage a service on a Linux instance, using a Tomcat application server as an example. The service resource works much the same way on Windows instances, although there are some differences in detail. For more information, see `service`.

> **Note**
>
> The example does a very minimal Tomcat installation, just enough to demonstrate the basics of how to use a `service` resource. For an example of how to implement recipes for a more functional Tomcat server, see Creating a Custom Tomcat Server Layer.

Topics

- Defining and Starting a Service
- Using notifies to Start or Restart a Service

Defining and Starting a Service

This section shows the basics of how to define and start a service.

To get started

1. In the `opsworks_cookbooks` directory, create a directory named `tomcat` and navigate to it.
2. Add a `metadata.rb` file to `tomcat` with the following content.

```
name "tomcat"
version "0.1.0"
```

3. Initialize and configure Test Kitchen, as described in Example 1: Installing Packages, and remove CentOS from the `platforms` list.
4. Add a `recipes` subdirectory to `tomcat`.

You use a `service` resource to manage a service. The following default recipe installs Tomcat and starts the service.

```
execute "install_updates" do
  command "apt-get update"
end

package "tomcat7" do
    action :install
end

include_recipe 'tomcat::service'

service 'tomcat' do
  action :start
end
```

The recipe does the following:

- The `execute` resource runs `apt-get update` to install the current system updates.

  For the Ubuntu instance used in this example, you must install the updates before installing Tomcat. Other systems might have different requirements.
- The `package` resource installs Tomcat 7.
- The included`tomcat::service` recipe defines the service and is discussed later.
- The `service` resource starts the Tomcat service.

  You can also use this resource to issue other commands, such as stopping and restarting the service.

The following example shows the `tomcat::service` recipe.

```
service 'tomcat' do
  service_name "tomcat7"
  supports :restart => true, :reload => false, :status => true
  action :nothing
end
```

This recipe creates the Tomcat service definition as follows:

- The resource name, `tomcat`, is used by other recipes to reference the service.

  For example, `default.rb` references `tomcat` to start the service.
- The `service_name` resource specifies the service name.

  When you list the services on the instance, the Tomcat service will be named tomcat7.
- `supports` specifies how Chef manages the service's `restart`, `reload`, and `status` commands.
  - `true` indicates that Chef can use the init script or other service provider to run the command.
  - `false` indicates that Chef must attempt to run the command by other means.

Notice that `action` is set to `:nothing`, which directs the resource to take no action. The service resource does support actions such as `start` and `restart`. However, this cookbook follows a standard practice of using a service definition that takes no action and starting or restarting the

service elsewhere. Each recipe that starts or restarts a service must first define it, so the simplest approach is to put the service definition in a separate recipe and include it in other recipes as needed.

> **Note**
>
> For simplicity, the default recipe for this example uses a `service` resource to start the service after running the service definition. A production implementation typically starts or restarts a service by using `notifies`, as discussed later.

To run the recipe

1. Create a `default.rb` file that contains the default recipe example and save it to `recipes`.
2. Create a `service.rb` file that contains the service definition example and save it to `recipes`.
3. Run `kitchen converge`, then log in to the instance and run the following command to verify that the service is running.

```
sudo service tomcat7 status
```

> **Note**
>
> If you were running `service.rb` separately from `default.rb`, you would have to edit `.kitchen.yml` to add `tomcat::service` to the run list. However, when you include a recipe, its code is incorporated into the parent recipe before the recipe is executed. `service.rb` is therefore basically a part of `default.rb` and doesn't require a separate run list entry.

Using notifies to Start or Restart a Service

Production implementations typically do not use `service` to start or restart a service. Instead, they add `notifies` to any of several resources. For example, if you want to restart the service after modifying a configuration file, you include `notifies` in the associated `template` resource. Using `notifies` has the following advantages over using a `service` resource to explicitly restart the service.

- The `notifies` element restarts the service only if the associated configuration file has changed, so there's no risk of causing an unnecessary service restart.
- Chef restarts the service at most once at the end of each run, regardless of how many `notifies` the run contains.

  For example, Chef run might include multiple template resources, each of which modifies a different configuration file and requires a service restart if the file has changed. However, you typically want to restart the service only once, at the end of the Chef run. Otherwise, you might attempt to restart a service that is not yet fully operational from an earlier restart, which can lead to errors.

This example modifies `tomcat::default` to include a `template` resource that uses `notifies` to restart the service. A realistic example would use a template resource that creates a customized version of one of the Tomcat configuration files, but those are rather long and complex. For simplicity, the example just uses the template resource from Creating a File from a Template. It doesn't have anything to do with Tomcat, but it provides a simple way to show how to use `notifies`. For an example of how to use templates to create Tomcat configuration files, see Setup Recipes.

To set up the cookbook

1. Add a `templates` subdirectory to `tomcat` and a `default` subdirectory to `templates`.
2. Copy the `example_data.json.erb` template from the `createfile` cookbook to the `templates/default` directory.
3. Add an `attributes` subdirectory to `tomcat`.
4. Copy the `default.rb` attribute file from the `createfile` cookbook to the `attributes` directory.

The following recipe uses `notifies` to restart the Tomcat service.

```
execute "install_updates" do
  command "apt-get update"
end

package "tomcat7" do
    action :install
```

```
end

include_recipe 'tomcat::service'

service 'tomcat' do
  action :enable
end

directory "/srv/www/shared" do
  mode 0755
  owner 'root'
  group 'root'
  recursive true
  action :create
end

template "/srv/www/shared/example_data.json" do
  source "example_data.json.erb"
  mode 0644
  variables(
    :a_boolean_var => true,
    :a_string_var => "some string"
  )
  only_if {node['createfile']['install_file']}
  notifies :restart, resources(:service => 'tomcat')
end
```

The example merges the recipe from Creating a File from a Template into the recipe from the preceding section, with two significant changes:

- The `service` resource is still there, but it now serves a somewhat different purpose.

  The `:enable` action enables the Tomcat service at boot.
- The template resource now includes `notifies`, which restarts the Tomcat service if `example_data.json` has changed.

  This ensures that the service is started when Tomcat is first installed and restarted after every configuration change.

To run the recipe

1. Run `kitchen destroy` to start with a clean instance.

2. Replace the code in `default.rb` with the preceding example.
3. Run `kitchen converge`, then log in to the instance and verify that the service is running.

> **Note**
>
> If you want to restart a service but the recipe doesn't include a resource such as `template` that supports `notifies`, you can instead use a dummy `execute` resource. For example
>
> ```
> execute 'trigger tomcat service restart' do
>   command 'bin/true'
>   notifies :restart, resources(:service => 'tomcat')
> end
> ```
>
> The `execute` resource must have a `command` attribute, even if you are using the resource only as a way to run `notifies`. This example gets around that requirement by running `/bin/true`, which is a shell command that simply returns a success code.

Example 9: Using Amazon EC2 Instances

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

To this point, you've been running instances locally in VirtualBox. While this is quick and easy, you will eventually want to test your recipes on an Amazon EC2 instance. In particular, if you want to run recipes on Amazon Linux, it is available only on Amazon EC2. You can use a similar system such as CentOS for preliminary implementation and testing, but the only way to fully test your recipes on Amazon Linux is with an Amazon EC2 instance.

This topic shows how to run recipes on an Amazon EC2 instance. You will use Test Kitchen and Vagrant in much the same way as the preceding sections, with two differences:

- The driver is `kitchen-ec2` instead of Vagrant.
- The cookbook's `.kitchen.yml` file must be configured with the information required to launch the Amazon EC2 instance.

> **Note**
>
> An alternative approach is to use the `vagrant-aws` Vagrant plug-in. For more information, see Vagrant AWS Provider.

You will need AWS credentials to create an Amazon EC2 instance. If you don't have an AWS account you can obtain one, as follows.

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

You should create an IAM user with permissions to access Amazon EC2 and save the user's access and secret keys to a secure location on your workstation. Test Kitchen will use those credentials to create the instance. The preferred way to provide credentials to Test Kitchen is to assign the keys to the following environment variables on your workstation.

> **Warning**
>
> IAM users have long-term credentials, which presents a security risk. To help mitigate this risk, we recommend that you provide these users with only the permissions they require to perform the task and that you remove these users when they are no longer needed.

- AWS_ACCESS_KEY – your user's access key, which will look something like AKIAIOSFODNN7EXAMPLE.
- AWS_SECRET_KEY – your user's secret key, which will look something like wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY.

This approach reduces the chances of accidentally compromising your account by, for example, uploading a project containing your credentials to a public repository.

To set up the cookbook

1. To use the `kitchen-ec2` driver, you must have the `ruby-dev` package installed on your system. The following example command shows how to use `aptitude` to install the package on a Ubuntu system.

   ```
   sudo aptitude install ruby1.9.1-dev
   ```

2. The `kitchen-ec2` driver is a gem, which you can install as follows:

   ```
   gem install kitchen-ec2
   ```

   Depending on your workstation, this command might require `sudo`, or you can also use a Ruby environment manager such as RVM. This procedure was tested with version 0.8.0 of

the `kitchen-ec2` driver, but there are newer versions. To install a specific version, run `gem install kitchen-ec2 -v <version number>`.

3. You must specify an Amazon EC2 SSH key pair that Test Kitchen can use to connect to the instance. If you don't have an Amazon EC2 key pair, see Amazon EC2 Key Pairs for information on how to create one. Note that the key pair must belong to the same AWS region as the instance. The example uses US West (N. California).

   After you have selected a key pair, create a subdirectory of `opsworks_cookbooks` named `ec2_keys` and copy the key pair's private key (`.pem`) file to that subdirectory. Note that putting the private key in `ec2_keys` is just a convenience that simplifies the code a bit; it can be anywhere on your system.

4. Create a subdirectory of `opsworks_cookbooks` named `createdir-ec2` and navigate to it.

5. Add a `metadata.rb` file to `createdir-ec2` with the following content.

```
name "createdir-ec2"
version "0.1.0"
```

6. Initialize Test Kitchen, as described in Example 1: Installing Packages. The following section describes how to configure `.kitchen.yml`, which is significantly more complicated for Amazon EC2 instances.

7. Add a `recipes` subdirectory to `createdir-ec2`.

Configuring .kitchen.yml for Amazon EC2

You configure `.kitchen.yml` with the information that the `kitchen-ec2` driver needs to launch an appropriately configured Amazon EC2 instance. The following is an example of a `.kitchen.yml` file for an Amazon Linux instance in the US West (N. California) region.

```
driver:
  name: ec2
  aws_ssh_key_id: US-East1
  region: us-west-1
  availability_zone: us-west-1c
  require_chef_omnibus: true
  security_group_ids: sg........
  subnet_id: subnet-.........
  associate_public_ip: true
```

```
  interface: dns

provisioner:
  name: chef_solo

platforms:
  -name: amazon
  driver:
    image_id: ami-xxxxxxxx
  transport:
    username: ec2-user
    ssh_key: ../ec2_keys/US-East1.pem

suites:
  - name: default
    run_list:
      - recipe[createdir-ec2::default]
    attributes:
```

You can use the default settings for the `provisioner` and `suites` sections, but you must modify the default `driver` and `platforms` settings. This example uses a minimal list of settings, and accepts the default values for the remainder. For a complete list of `kitchen-ec2` settings, see Kitchen::Ec2: A Test Kitchen Driver for Amazon EC2.

The example sets the following `driver` attributes. It assumes that you have assigned your user's access and secret keys to the standard environment variables, as discussed earlier. The driver uses those keys by default. Otherwise, you must explicitly specify the keys by adding `aws_access_key_id` and `aws_secret_access_key` to the `driver` attributes, set to the appropriate key values.

name

(Required) This attribute must be set to `ec2`.

aws_ssh_key_id

(Required) The Amazon EC2 SSH key pair name, which is named `US-East1` in this example.

transport.ssh_key

(Required) The private key (`.pem`) file for the key that you specified for `aws_ssh_key_id`. For this example, the file is named `US-East1.pem` and is in the `../opsworks/ec2_keys` directory.

region

(Required) The instance's AWS region. The example uses US West (N. California), which is represented by `us-west-1`).

availability_zone

(Optional) The instance's Availability Zone. If you omit this setting, Test Kitchen uses a default Availability Zone for the specified region, which is `us-west-1b` for US West (N. California). However, the default zone might not be available for your account. In that case, you must explicitly specify an Availability Zone. As it happens, the account used to prepare the examples doesn't support `us-west-1b`, so the example explicitly specifies `us-west-1c`.

require_chef_omnibus

When set to `true`, this setting ensures that the omnibus installer is used to install `chef-client` to all platform instances.

security_group_ids

(Optional) A list of security group IDs to apply to the instance. This setting applies the `default` security group to the instance. Make sure that the security group ingress rules allow inbound SSH connections, or Test Kitchen will not be able to communicate with the instance. If you use the `default` security group, you might need to edit it accordingly. For more information, see Amazon EC2 Security Groups.

subnet_id

The ID of the target subnet for the instance, if applicable.

associate_public_ip

You can have Amazon EC2 associate a public IP address with the instance if you want to be able to access the instance from the Internet.

interface

The host name configuration type that you use to access the instance. Valid values are `dns`, `public`, `private`, or `private_dns`. If you do not specify a value for this attribute, `kitchen-ec2` sets up the host name configuration in the following order. If you omit this attribute, the configuration type is not set.

1. DNS name
2. Public IP address

3. Private IP address
4. Private DNS name

> **Important**
>
> Rather than use your account credentials for the access and secret keys, you should create a user and provide those credentials to Test Kitchen. For more information, see Best Practices for Managing AWS Access Keys.
> Be careful not to put `.kitchen.yml` in a publicly accessible location, such as uploading it to a public GitHub or Bitbucket repository. Doing so exposes your credentials and could compromise your account's security.

The `kitchen-ec2` driver provides default support for the following platforms:

- ubuntu-10.04
- ubuntu-12.04
- ubuntu-12.10
- ubuntu-13.04
- ubuntu-13.10
- ubuntu-14.04
- centos-6.4
- debian-7.1.0
- windows-2012r2
- windows-2008r2

If you want to use one or more of these platforms, add the appropriate platform names to `platforms`. The `kitchen-ec2` driver automatically selects an appropriate AMI and generates an SSH user name. You can use other platforms—this example uses Amazon Linux—but you must explicitly specify the following `platforms` attributes.

name

The platform name. This example uses Amazon Linux, so `name` is set to `amazon`.

driver

The `driver` attributes, which include the following:

- `image_id` – The platform's AMI, which must belong to the specified region. The example uses `ami-ed8e9284`, an Amazon Linux AMI from the US West (N. California) region.
- `transport.username` – The SSH user name that Test Kitchen will use to communicate with the instance.

  Use `ec2-user` for Amazon Linux. Other AMIs might have different user names.

Replace the code in `.kitchen.yml` with the example, and assign appropriate values to account-specific attributes such as `aws_access_key_id`.

Running the Recipe

This example uses the recipe from Iteration.

To run the recipe

1. Create a file named `default.rb` with the following code and save it to the cookbook's `recipes` folder.

   ```
   directory "/srv/www/shared" do
     mode 0755
     owner 'root'
     group 'root'
     recursive true
     action :create
   end
   ```

2. Run `kitchen converge` to execute the recipe. Note that this command will take longer to complete than the previous examples because of the time required to launch and initialize an Amazon EC2 instance.
3. Go to the Amazon EC2 console, select the US West (N. California)) region, and click **Instances** in the navigation pane. You will see the newly created instance in the list.
4. Run `kitchen login` to log in to the instance, just as you have been doing for instances running in VirtualBox. You will see the newly created directories under `/srv`. You can also use your favorite SSH client to connect to the instance.

Next Steps

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This chapter walked you through the basics of how to implement Chef cookbooks, but there's much more:

- The examples showed you how to use some of the more commonly used resources, but there are many more.

  For the resources that were covered, the examples used only some of the available attributes and actions. For a complete reference, see About Resources and Providers.
- The examples used only the core cookbook elements: `recipes`, `attributes`, `files`, and `templates`.

  Cookbooks can also include a variety of other elements, such as `libraries`, `definitions`, and `specs`. For more information, see the Chef documentation.
- The examples used Test Kitchen only as a convenient way to start instances, run recipes, and log in to instances.

  Test Kitchen is primarily a testing platform that you can use to run a variety of tests on your recipes. If you haven't done so already, go through the rest of the Test Kitchen walkthrough, which introduces you to its testing features.
- Implementing Cookbooks for OpsWorks Stacks provides some more advanced examples, and shows how to implement cookbooks for OpsWorks Stacks.

Implementing Cookbooks for OpsWorks Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Cookbook Basics introduced you to cookbooks and recipes. The examples in that section were simple by design and will work on any instance that supports Chef, including OpsWorks Stacks instances. To implement more sophisticated cookbooks for OpsWorks Stacks, you typically need to take full advantage of the OpsWorks Stacks environment, which differs from standard Chef in a number of ways.

This topic describes the basics of implementing recipes for OpsWorks Stacks instances.

> **Note**
>
> If you are not familiar with how to implement cookbooks, you should start with Cookbook Basics.

Topics

- Running a Recipe on an OpsWorks Stacks Linux Instance
- Running a Recipe on a Windows Instance
- Running a Windows PowerShell Script
- Mocking the Stack Configuration and Deployment Attributes on Vagrant
- Using Stack Configuration and Deployment Attribute Values
- Using an External Cookbook on a Linux Instance: Berkshelf
- Using the SDK for Ruby: Downloading Files from Amazon S3
- Installing Windows Software
- Overriding Built-In Attributes
- Overriding Built-In Templates

Running a Recipe on an OpsWorks Stacks Linux Instance

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Test Kitchen and Vagrant provide a simple and efficient way to implement cookbooks, but to verify that a cookbook's recipes will run correctly in production, you must run them on an OpsWorks Stacks instance. This topic describes how to install a custom cookbook on an OpsWorks Stacks Linux instance and run a simple recipe. The topic also provides some tips for efficiently fixing recipe bugs.

For a description of how to run recipes on Windows instances, see Running a Recipe on a Windows Instance.

Topics

- Creating and Running the Recipe
- Executing the Recipe Automatically
- Troubleshooting and Fixing Recipes

Creating and Running the Recipe

First, you need to create a stack. The following briefly summarizes how to create a stack for this example. For more information, see Create a New Stack.

To create a stack

1. Open the OpsWorks Stacks console and click **Add Stack**.
2. Specify the following settings, accept the defaults for the other settings, and click **Add Stack**.
   - **Name** – OpsTest
   - **Default SSH key** – An Amazon EC2 key pair

If you need to create an Amazon EC2 key pair, see Amazon EC2 Key Pairs. Note that the key pair must belong to the same AWS region as the instance. The example uses the default US West (Oregon) region.

3. Click **Add a layer** and add a custom layer to the stack with the following settings.

   - **Name** – OpsTest
   - **Short name** – opstest

   Any layer type will actually work for Linux stacks, but the example doesn't require any of the packages that are installed by the other layer types, so a custom layer is the simplest approach.

4. Add a 24/7 instance with default settings to the layer and start it.

While the instance is starting up—it usually takes several minutes—you can create the cookbook. This example will use a slightly modified version of the recipe from Conditional Logic, which creates a data directory whose name depends on the platform.

To set up the cookbook

1. Create a directory within `opsworks_cookbooks` named `opstest` and navigate to it.
2. Create a `metadata.rb` file with the following content and save it to `opstest`.

```
name "opstest"
version "0.1.0"
```

3. Create a `recipes` directory within `opstest`.
4. Create a `default.rb` file with the following recipe and save it to the `recipes` directory.

```
Chef::Log.info("******Creating a data directory.******")

data_dir = value_for_platform(
  "centos" => { "default" => "/srv/www/shared" },
  "ubuntu" => { "default" => "/srv/www/data" },
  "default" => "/srv/www/config"
)
```

```
directory data_dir do
  mode 0755
  owner 'root'
  group 'root'
  recursive true
  action :create
end
```

Notice that the recipe logs a message, but it does so by calling `Chef::Log.info`. You aren't using Test Kitchen for this example, so the `log` method isn't very useful. `Chef::Log.info` puts the message into the Chef log, which you can read after the Chef run is finished. OpsWorks Stacks provides an easy way to view these logs, as described later.

> **Note**
>
> Chef logs usually contain a lot of routine and relatively uninteresting information. The '*' characters bracketing the message text make it easier to spot.

5. Create a `.zip` archive of `opsworks_cookbooks`. To install your cookbook on an OpsWorks Stacks instance, you must store it in a repository and provide OpsWorks Stacks with the information required to download the cookbook to the instance. You can store your cookbooks in any of several supported repository types. This example stores an archive file containing the cookbooks in an Amazon S3 bucket. For more information on cookbook repositories, see Cookbook Repositories.

   > **Note**
   >
   > For simplicity, this example just archives the entire `opsworks_cookbooks` directory. However, it means that OpsWorks Stacks will download all the cookbooks in `opsworks_cookbooks` to the instance, even though you will use only one of them. To install only the example cookbook, create another parent directory and move `opstest` to that directory. Then create a `.zip` archive of the parent directory and use it instead of `opsworks_cookbooks.zip`.
   >
   > Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

6. Upload the archive to an Amazon S3 bucket, make the archive public, and record the archive's URL.

You can now install the cookbook and run the recipe.

To run the recipe

1. Edit the stack to enable custom cookbooks, and specify the following settings.

   - **Repository type** – **S3 Archive**
   - **Repository URL** – The cookbook archive URL that you recorded earlier

   Use the default values for the other settings and click **Save** to update the stack configuration.
2. Run the Update Custom Cookbooks stack command, which installs the current version of your custom cookbooks on the stack's instances. If an earlier version of your cookbooks is present, this command overwrites it.
3. Execute the recipe by running the **Execute Recipes** stack command with **Recipes to execute** set to **`opstest::default`**. This command initiates a Chef run, with a run list that consists of `opstest::default`.

After the recipe runs successfully, you can verify it.

To verify opstest

1. The first step is to examine the Chef log. Click **show** in the opstest1 instance's **Log** column to display the log. Scroll down and you will see your log message near the bottom.

```
...
[2014-07-31T17:01:45+00:00] INFO: Storing updated cookbooks/opsworks_cleanup/
attributes/customize.rb in the cache.
[2014-07-31T17:01:45+00:00] INFO: Storing updated cookbooks/opsworks_cleanup/
metadata.rb in the cache.
[2014-07-31T17:01:46+00:00] INFO: ******Creating a data directory.******
[2014-07-31T17:01:46+00:00] INFO: Processing template[/etc/hosts] action create
 (opsworks_stack_state_sync::hosts line 3)
...
```

2. Use SSH to log in to the instance and list the contents of `/srv/www/`.

If you followed all the steps, you will see `/srv/www/config` rather than the `/srv/www/shared` directory you were expecting. The following section provides some guidelines for quickly fixing such bugs.

Executing the Recipe Automatically

The **Execute Recipes** command is a convenient way to test custom recipes, which is why it is used in most of these examples. However, in practice you typically run recipes at standard points in an instance's lifecycle, such as after the instance finishes booting or when you deploy an app. OpsWorks Stacks simplifies running recipes on your instance by supporting a set of lifecycle events for each layer: Setup, Configure, Deploy, Undeploy, and Shutdown. You can have OpsWorks Stacks run a recipe automatically on a layer's instances by assigning the recipe to the appropriate lifecycle event.

You would typically create directories as soon as an instance finishes booting, which corresponds to the Setup event. The following shows how to run the example recipe at setup, using the same stack that you created earlier in the example. You can use the same procedure for the other events.

To automatically run a recipe at setup

1. Choose **Layers** in the navigation pane and then chose the pencil icon next to the OpsTest layer's **Recipes** link.
2. Add **`opstest::default`** to the layer's **Setup** recipes, click **+** to add it to the layer, and choose **Save** to save the configuration.
3. Choose **Instances**, add another instance to the layer, and start it.

   The instance should be named `opstest2`. After it finishes booting, OpsWorks Stacks will run `opstest::default`.
4. After the `opstest2` instance is online, verify that `/srv/www/shared` is present.

> **Note**
>
> If you have assigned recipes to the Setup, Configure, or Deploy events, you also run them manually by using a stack command (Setup and Configure) or a deploy command (Deploy)

to trigger the event. Note that if you have multiple recipes assigned to an event, these commands run all of them.

Troubleshooting and Fixing Recipes

If you aren't getting the expected results, or your recipes don't even run successfully, troubleshooting typically starts by examining the Chef log. It contains a detailed description of the run and includes any inline log messages from your recipes. The logs are particularly useful if your recipe simply failed. When that happens, Chef logs the error, including a stack trace.

If the recipe was successful, as it was for this example, the Chef log often isn't much help. In this case, you can figure out the problem by just taking a closer look at the recipe, the first few lines in particular:

```
Chef::Log.info("******Creating a data directory.******")

data_dir = value_for_platform(
  "centos" => { "default" => "/srv/www/shared" },
  "ubuntu" => { "default" => "/srv/www/data" },
  "default" => "/srv/www/config"
)
...
```

CentOS is a reasonable stand-in for Amazon Linux when you are testing recipes on Vagrant, but now you are running on an actual Amazon Linux instance. The platform value for Amazon Linux is `amazon`, which isn't included in the `value_for_platform` call, so the recipe creates `/srv/www/config` by default. For more information on troubleshooting, see Debugging and Troubleshooting Guide.

Now that you have identified the problem, you need to update the recipe and verify the fix. You could go back to the original source files, update `default.rb`, upload a new archive to Amazon S3, and so on. However, that process can be a bit tedious and time consuming. The following shows a much quicker approach that is especially useful for simple recipe bugs like the one in the example: edit the recipe on the instance.

To edit a recipe on an instance

1. Use SSH to log in to the instance and then run `sudo su` to elevate your privileges. You need root privileges to access the cookbook directories.
2. OpsWorks Stacks stores your cookbook in `/opt/aws/opsworks/current/site-cookbooks`, so navigate to `/opt/aws/opsworks/current/site-cookbooks/opstest/recipes`.

   > **Note**
   >
   > OpsWorks Stacks also stores a copy of your cookbooks in `/opt/aws/opsworks/current/merged-cookbooks`. Don't edit that cookbook. When you execute the recipe, OpsWorks Stacks copies the cookbook from `.../site-cookbooks` to `.../merged-cookbooks`, so any changes you make in `.../merged-cookbooks` will be overwritten.

3. Use a text editor on the instance to edit `default.rb`, and replace `centos` with `amazon`. Your recipe should now look like the following.

```
Chef::Log.info("******Creating a data directory.******")

data_dir = value_for_platform(
  "amazon" => { "default" => "/srv/www/shared" },
  "ubuntu" => { "default" => "/srv/www/data" },
  "default" => "/srv/www/config"
)
...
```

To verify the fix, execute the recipe by running the **Execute Recipe** stack command again. The instance should now have a `/srv/www/shared` directory. If you need to make further changes to a recipe, you can run **Execute Recipe** as often as you like; you don't need to stop and restart the instance each time you run the command. When you are satisfied that the recipe is working correctly, don't forget to update the code in your source cookbook.

Note

If you have assigned your recipe to a lifecycle event so OpsWorks Stacks runs it automatically, you can always use **Execute Recipe** to rerun the recipe. You can also rerun the recipe as many times as you want without restarting the instance by using the OpsWorks Stacks console to manually trigger the appropriate event. However, this approach runs all of the event's recipes. Here's a reminder:

- Use a stack command to trigger Setup or Configure events.
- Use a deploy command to trigger Deploy or Undeploy events.

Running a Recipe on a Windows Instance

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This topic is basically an abbreviated version of Running a Recipe on a Linux Instance, which shows you how to run a recipe on a Windows stack. We recommend that you go through Running a Recipe on a Linux Instance first, because it provides a more detailed discussion, most of which is relevant to either type of operating system.

For a description of how to run recipes on OpsWorks Stacks Linux instances, see Running a Recipe on a Linux Instance.

Topics

- Enabling RDP Access
- Creating and Running the Recipe
- Executing the Recipe Automatically

Enabling RDP Access

Before you start, if you have not done so already, you must set up a security group with an inbound rule that allows RDP access for your instances . You will need that group when you create the stack.

When you create the first stack in a region, OpsWorks Stacks creates a set of security groups. They include one named something like `AWS-OpsWorks-RDP-Server`, which OpsWorks Stacks attaches to all Windows instances to allow RDP access. However, by default, this security group does not have any rules, so you must add an inbound rule to allow RDP access to your instances.

To allow RDP access

1. Open the Amazon EC2 console, set it to the stack's region, and choose **Security Groups** from the navigation pane.
2. Choose **AWS-OpsWorks-RDP-Server**, choose the **Inbound** tab, and choose **Edit**.
3. Add a rule with the following settings:

   - **Type** – **RDP**
   - **Source** – The permissible source IP addresses.

     You typically allow inbound RDP requests from your IP address or a specified IP address range (typically your corporate IP address range).

> **Note**
>
> As described later, you also must edit user permissions to authorize RDP access for regular users.

For more information, see Logging In with RDP.

Creating and Running the Recipe

The following briefly summarizes how to create a stack for this example. For more information, see Create a New Stack.

Create a stack

1. Open the OpsWorks Stacks console and choose **Add Stack**. Specify the following settings, accept the defaults for the other settings, and choose **Add Stack**.

- **Name** – WindowsRecipeTest
- **Region** – US West (Oregon)

  This example will work in any region, but we recommend using US West (Oregon) for tutorials.
- **Default operating system** – Microsoft Windows Server 2012 R2

2. Choose **Add a layer** and add a custom layer to the stack with the following settings.

   - **Name** – RecipeTest
   - **Short name** – recipetest

3. Add a 24/7 instance with default settings to the RecipeTest layer and start it.

   OpsWorks Stacks automatically assigns `AWS-OpsWorks-RDP-Server` to this instance, which allows authorized users to log in to the instance.

4. Choose **Permissions** and then **Edit**, and choose **SSH/RDP** and **sudo/admin**. Regular users need this authorization in addition to the `AWS-OpsWorks-RDP-Server` security group to log in to the instance.

   > **Note**
   >
   > You can also log in as Administrator, but it requires a different procedure. For more information, see Logging In with RDP.

While the instance is starting up—it usually takes several minutes—you can create the cookbook. The recipe for this example creates a data directory, and is basically the recipe from Example 3: Creating Directories, modified for Windows.

> **Note**
>
> When implementing cookbooks for OpsWorks Stacks Windows instances, you use a somewhat different directory structure than you do when implementing cookbooks for OpsWorks Stacks Linux instances. For more information, see Cookbook Repositories.

To set up the cookbook

1. Create a directory named `windowstest` and navigate to it.
2. Create a `metadata.rb` file with the following content and save it to `windowstest`.

   ```
   name "windowstest"
   version "0.1.0"
   ```

3. Create a `recipes` directory within `windowstest`.
4. Create a `default.rb` file with the following recipe and save it to the `recipes` directory.

   ```
   Chef::Log.info("******Creating a data directory.******")

   directory 'C:\data' do
     rights :full_control, 'instance_name\username'
     inherits false
     action :create
   end
   ```

   Replace *username* with your user name.

5. Put the cookbook in a repository.

   To install your cookbook on an OpsWorks Stacks instance, you must store it in a repository and provide OpsWorks Stacks with the information required to download the cookbook to the instance. You can store Windows cookbooks as an archive file in an S3 bucket or in a Git repository. This example uses an S3 bucket, so you must create a .zip archive of the `windowstest` directory. For more information on cookbook repositories, see Cookbook Repositories.

6. Upload the archive to an S3 bucket, make the archive public, and record the archive's URL. You can also use a private archive, but a public archive is sufficient for this example and somewhat easier to work with.

   Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

You can now install the cookbook and run the recipe.

To run the recipe

1. Edit the stack to enable custom cookbooks and specify the following settings.

   - **Repository type** – **S3 Archive**
   - **Repository URL** – The cookbook archive URL that you recorded earlier

   Accept the default values for the other settings and choose **Save** to update the stack configuration.
2. Run the Update Custom Cookbooks stack command, which installs the current version of your custom cookbooks on the stack's instances, including online instances. If an earlier version of your cookbooks is present, this command overwrites it.
3. After Update Custom Cookbooks is finished, execute the recipe by running the **Execute Recipes** stack command with **Recipes to execute** set to **`windowstest::default`**. This command initiates a Chef run, with a run list that consists of your recipe.

After the recipe runs successfully, you can verify it.

To verify windowstest

1. Examine the Chef log. Choose **show** in the opstest1 instance's **Log** column to display the log. Scroll down and you will see your log message near the bottom.

   ```
   ...
   [2014-07-31T17:01:45+00:00] INFO: Storing updated cookbooks/opsworks_cleanup/
   attributes/customize.rb in the cache.
   [2014-07-31T17:01:45+00:00] INFO: Storing updated cookbooks/opsworks_cleanup/
   metadata.rb in the cache.
   [2014-07-31T17:01:46+00:00] INFO: ******Creating a data directory.******
   [2014-07-31T17:01:46+00:00] INFO: Processing template[/etc/hosts] action create
    (opsworks_stack_state_sync::hosts line 3)
   ...
   ```
2. Choose **Instances**, choose **rdp** in the instance's **Actions** column, and request an RDP password with a suitable expiration time. Copy the DNS name, user name, and password. You can then can use that information with an RDP client, such as the Windows Remote Desktop Connection

client, to log in to the instance and verify that `c:\data` exists. For more information, see Logging In with RDP.

> **Note**
>
> If your recipe isn't working properly, see Troubleshooting and Fixing Recipes for troubleshooting tips; most of them also apply to Windows instances. If you want to test your fix by editing the recipe on the instance, look for your cookbook in the `C:\chef\cookbooks` directory, where OpsWorks Stacks installs custom cookbooks.

Executing the Recipe Automatically

The **Execute Recipes** command is a convenient way to test custom recipes, which is why it is used in most of these examples. However, in practice you typically run recipes at standard points in an instance's lifecycle, such as after the instance finishes booting or when you deploy an app. OpsWorks Stacks simplifies running recipes on your instance by supporting a set of lifecycle events for each layer: Setup, Configure, Deploy, Undeploy, and Shutdown. You can have OpsWorks Stacks run a recipe automatically on a layer's instances by assigning the recipe to the appropriate lifecycle event.

You would typically create directories as soon as an instance finishes booting, which corresponds to the Setup event. The following shows how to run the example recipe at setup, using the same stack that you created earlier in the example. You can use the same procedure for the other events.

To automatically run a recipe at setup

1. Choose **Layers** in the navigation pane and then choose the pencil icon next to the RecipeTest layer's **Recipes** link.
2. Add **`windowstest::default`** to the layer's **Setup** recipes, choose **+** to add it to the layer, and choose **Save** to save the configuration.
3. Choose **Instances**, add another instance to the layer, and start it.

   The instance should be named `recipetest2`. After it finishes booting, OpsWorks Stacks will run `windowstest::default`.
4. After the `recipetest2` instance is online, verify that `c:\data` is present.

> **Note**
>
> If you have assigned recipes to the Setup, Configure, or Deploy events, you can also run them manually by using a stack command (Setup and Configure) or a deploy command (Deploy) to trigger the event. Note that if you have multiple recipes assigned to an event, these commands run all of them.

Running a Windows PowerShell Script

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> These examples assume that you have already done the Running a Recipe on a Windows Instance example. If not, you should do that example first. In particular, it describes how to enable RDP access to your instances.

One way to have a recipe perform tasks on a Windows instance—especially tasks that do not have a corresponding Chef resource—is to have the recipe run a Windows PowerShell script. This section introduces you to the basics by describing how to use a Windows PowerShell script to install a Windows feature.

The `powershell_script` resource runs Windows PowerShell cmdlets on an instance. The following example uses a Install-WindowsFeature cmdlet to install an XPS viewer on the instance.

The following briefly summarizes how to create a stack for this example. For more information, see Create a New Stack.

Create a stack

1. Open the OpsWorks Stacks console and choose **Add Stack**. Specify the following settings, accept the defaults for the other settings, and click **Add Stack**.

   - **Name** – PowerShellTest
   - **Region** – US West (Oregon)

     This example will work in any region, but we recommend using US West (Oregon) for tutorials.
   - **Default operating system** – Microsoft Windows Server 2012 R2
2. Choose **Add a layer** and add a custom layer to the stack with the following settings.

   - **Name** – PowerShell
   - **Short name** – powershell
3. Add a 24/7 instance to with default settings to the PowerShell layer and start it.
4. Choose **Permissions** and then **Edit**, and select **SSH/RDP** and **sudo/admin**. You need this authorization in addition to the `AWS-OpsWorks-RDP-Server` security group to log in to the instance as a regular user.

While the instance is starting up—it usually takes several minutes—you can create the cookbook. The recipe for this example creates a data directory, and is basically the recipe from Example 3: Creating Directories, modified for Windows.

To set up the cookbook

1. Create a directory named `powershell` and navigate to it.
2. Create a `metadata.rb` file with the following content and save it to `windowstest`.

   ```
   name "powershell"
   version "0.1.0"
   ```
3. Create a `recipes` directory within the `powershell` directory.
4. Create a `default.rb` file with the following recipe and save it to the `recipes` directory.

   ```
   Chef::Log.info("******Installing XPS.******")
   ```

```
powershell_script "Install XPS Viewer" do
  code <<-EOH
    Install-WindowsFeature XPS-Viewer
  EOH
  guard_interpreter :powershell_script
  not_if "(Get-WindowsFeature -Name XPS-Viewer).installed"
end
```

- The `powershell_script` resource runs a cmdlet to install the XPS viewer.

  This example runs only one cmdlet, but the `code` block can contain any number of command lines.
- The `guard_interpreter` attribute directs Chef to use the 64-bit version of Windows PowerShell.
- The `not_if` guard attribute ensures that Chef does not install the feature if it has already been installed.

5. Create a `.zip` archive of the `powershell` directory.
6. Upload the archive to an Amazon S3 bucket, make the archive public, and record the archive's URL. You can also use a private archive, but a public archive is sufficient for this example, and somewhat easier to work with.

   Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

You can now install the cookbook and run the recipe.

To run the recipe

1. Edit the stack to enable custom cookbooks and specify the following settings.

   - **Repository type** – **S3 Archive**
   - **Repository URL** – The cookbook archive URL that you recorded earlier

   Accept the default values for the other settings and choose **Save** to update the stack configuration.

2. Run the **Update Custom Cookbooks** stack command to install the current version of your custom cookbooks on the instance.
3. After **Update Custom Cookbooks** has finished, execute the recipe by running the **Execute Recipes** stack command with **Recipes to execute** set to **`powershell::default`**.

> **Note**
>
> This example uses **Execute Recipes** for convenience, but you typically have OpsWorks Stacks run your recipes automatically by assigning them to the appropriate lifecycle event. You can run such recipes by manually triggering the event. You can use a stack command to trigger Setup and Configure events, and a deploy command to trigger Deploy and Undeploy events.

After the recipe runs successfully, you can verify it.

To verify the powershell recipe

1. Examine the Chef log. Click **show** in the powershell1 instance's **Log** column to display the log. Scroll down and you will see your log message near the bottom.

```
...
[2015-04-27T18:12:09+00:00] INFO: Storing updated cookbooks/powershell/metadata.rb
 in the cache.
[2015-04-27T18:12:09+00:00] INFO: ******Installing XPS.******
[2015-04-27T18:12:09+00:00] INFO: Processing powershell_script[Install XPS Viewer]
 action run (powershell::default line 3)
[2015-04-27T18:12:09+00:00] INFO: Processing powershell_script[Guard resource]
 action run (dynamically defined)
[2015-04-27T18:12:42+00:00] INFO: powershell_script[Install XPS Viewer] ran
 successfully
...
```

2. Use RDP to log in to the instance and open the **Start** menu. XPS Viewer should be listed with **Windows Accessories**.

Mocking the Stack Configuration and Deployment Attributes on Vagrant

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

This topic applies only to Linux instances. Test Kitchen does not yet support Windows, so you will run all Windows examples on OpsWorks Stacks instances.

OpsWorks Stacks adds stack configuration and deployment attributes to the node object for each instance in your stack for every lifecycle event. These attributes provide a snapshot of the stack configuration, including the configuration of each layer and its online instances, the configuration of each deployed app, and so on. Because these attributes are in the node object, they can be accessed by any recipe; most recipes for OpsWorks Stacks instances use one or more of these attributes.

An instance running in a Vagrant box is not managed by OpsWorks Stacks, so its node object does not include any stack configuration and deployment attributes by default. However, you can add a suitable set of attributes to the Test Kitchen environment. Test Kitchen then adds the attributes to the instance's node object, and your recipes can access the attributes much like they would on an OpsWorks Stacks instance.

This topic shows how to obtain a copy of a suitable stack configuration and deployment attributes, install the attributes on an instance, and access them.

Note

If you are using Test Kitchen to run tests on your recipes, fauxhai provides an alternative way to mock stack configuration and deployment JSON.

To set up the cookbook

1. Create a subdirectory of `opsworks_cookbooks` named `printjson` and navigate to it.
2. Initialize and configure Test Kitchen, as described in Example 1: Installing Packages.
3. Add two subdirectories to `printjson`: `recipes` and `environments`.

You could mock stack configuration and deployment attributes by adding an attribute file to your cookbook with the appropriate definitions, but a better approach is to use the Test Kitchen environment. There are two basic approaches:

- Add attribute definitions to `.kitchen.yml`.

  This approach is most useful if you have just a few attributes. For more information, see kitchen.yml.
- Define the attributes in an environment file and reference the file in `.kitchen.yml`.

  This approach is usually preferable for stack configuration and deployment attributes because the environment file is already in JSON format. You can get a copy of the attributes in JSON format from a suitable OpsWorks Stacks instance and just paste it in. All of the examples use an environment file.

The simplest way to create a stack configuration and deployment attributes for your cookbook is to create an appropriately configured stack and copy the resulting attributes from an instance as JSON. To keep your Test Kitchen environment file manageable, you can then edit that JSON to have only the attributes that your recipes need. The examples in this chapter are based on the stack from Getting Started with Chef 11 Linux Stacks, which is a simple PHP application server stack with a load balancer, PHP application servers, and a MySQL database server.

To create a stack configuration and deployment JSON

1. Create MyStack as described in Getting Started with Chef 11 Linux Stacks, including deploying SimplePHPApp. If you prefer, you can omit the second PHP App Server instance called for in Step 4: Scale Out MyStack; the examples don't use those attributes.
2. If you haven't already done so, start the `php-app1` instance, and then log in with SSH.
3. In the terminal window, run the following agent cli command:

```
sudo opsworks-agent-cli get_json
```

This command prints the instance's most recent stack configuration and deployment attributes to the terminal window in JSON format.

4. Copy the JSON to a `.json` file and save it in a convenient location on your workstation. The details depend on your SSH client. For example, if you are using PuTTY on Windows, you can run the `Copy All to Clipboard` command, which copies all the text in the terminal window to the Windows clipboard. You can then paste the contents into a `.json` file and edit the file to remove extraneous text.
5. Edit MyStack JSON as needed. Stack configuration and deployment attributes are numerous, and cookbooks typically use only a small subset of them. To keep your environment file manageable, you can edit the JSON so that it retains the original structure but contains only the attributes that your cookbooks actually use.

   This example uses a heavily edited version of the MyStack JSON that includes just two `['opsworks']['stack']` attributes, `['id]` and `['name']`. Create an edited version of the MyStack JSON that looks something like the following:

```
{
  "opsworks": {
    "stack": {
      "name": "MyStack",
      "id": "42dfd151-6766-4f1c-9940-ba79e5220b58",
    },
  },
}
```

To get this JSON into the instance's node object, you need to add it to a Test Kitchen environment.

To add stack configuration and deployment attributes to the Test Kitchen environment

1. Create an environment file named `test.json` with the following contents and save it to the cookbook's `environments` folder.

```
{
```

```
  "default_attributes": {
    "opsworks" : {
      "stack" : {
        "name" : "MyStack",
        "id" : "42dfd151-6766-4f1c-9940-ba79e5220b58"
      }
    }
  },
  "chef_type" : "environment",
  "json_class" : "Chef::Environment"
}
```

The environment file has the following elements:

- `default_attributes` – The default attributes in JSON format.

  These attributes are added to the node object with the `default` attribute type, which is the type used by all of the stack configuration and deployment JSON attributes. This example uses the edited version of the stack configuration and deployment JSON shown earlier.
- `chef_type` – Set this element to `environment`.
- `json_class` – Set this element to `Chef::Environment`.

2. Edit `.kitchen.yml` to define the Test Kitchen environment, as follows.

```
---
driver:
  name: vagrant

provisioner:
  name: chef_solo
  environments_path: ./environments

platforms:
  - name: ubuntu-12.04

suites:
  - name: printjson
    provisioner:
      solo_rb:
        environment: test
    run_list:
```

```
      - recipe[printjson::default]
    attributes:
```

You define the environment by adding the following elements to the default `.kitchen.yml` created by `kitchen init`.

provisioner

Add the following elements.

- `name` – Set this element to `chef_solo`.

  To replicate the OpsWorks Stacks environment more closely, you could use Chef client local mode instead of Chef solo. Local mode is a Chef client option that uses a lightweight version of Chef server (Chef Zero) that runs locally on the instance instead of a remote server. It enables your recipes to use Chef server features such as search or data bags without connecting to a remote server.

- `environments_path` – The cookbook subdirectory that contains the environment file, `./environments` for this example.

suites:provisioner

Add a `solo_rb` element with an `environment` element set to the environment file's name, minus the .json extension. This example sets `environment` to `test`.

3. Create a recipe file named `default.rb` with the following content and save it to the cookbook's `recipes` directory.

```
log "Stack name: #{node['opsworks']['stack']['name']}"
log "Stack id: #{node['opsworks']['stack']['id']}"
```

This recipe simply logs the two stack configuration and deployment values that you added to the environment. Although the recipe is running locally in Virtual Box, you reference those attributes using the same node syntax that you would if the recipe were running on an OpsWorks Stacks instance.

4. Run `kitchen converge`. You should see something like the following log output.

```
...
Converging 2 resources
```

```
Recipe: printjson::default
  * log[Stack name: MyStack] action write[2014-07-01T23:14:09+00:00] INFO:
 Processing log[Stack name: MyStack] action write (printjson::default line 1)

[2014-07-01T23:14:09+00:00] INFO: Stack name: MyStack

  * log[Stack id: 42dfd151-6766-4f1c-9940-ba79e5220b58] action
 write[2014-07-01T23:14:09+00:00] INFO: Processing log[Stack id:
 42dfd151-6766-4f1c-9940-ba79e5220b58] action write (printjson::default line 2)

[2014-07-01T23:14:09+00:00] INFO: Stack id: 42dfd151-6766-4f1c-9940-ba79e5220b58

...
```

Using Stack Configuration and Deployment Attribute Values

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Recipes often need information about the stack configuration or deployed apps. For example, you might need a list of the stack's IP addresses to create a configuration file, or an app's deployment directory to create a log directory. Instead of storing this data on a central server, OpsWorks Stacks installs a set of stack configuration and deployment attributes in each instance's node object for each lifecycle event. These attributes represent the current stack state, including deployed apps. Recipes can then obtain the data they need from the node object.

Note

Applications sometimes need information from the node object, such as stack configuration and deployment attribute values. However, an application cannot access the node object. To provide node object data to an application, you can implement a recipe that retrieves the required information from the node object and puts it in a file

in a convenient format. The application can then read the data from the file. For more information and an example, see Passing Data to Applications.

Recipes can obtain stack configuration and deployment attribute values from the node object as follows.

- Directly, by using an attribute's fully qualified name.

  You can use this approach with any Linux stack, but not with Windows stacks.
- With Chef search, which you can use to query the node object for attribute values.

  You can use this approach with Windows stacks and Chef 11.10 Linux stacks.

> **Note**
>
> With Linux stacks, you can use the agent CLI to get a copy of an instance's stack configuration and deployment attributes in JSON format. For more information, see Mocking the Stack Configuration and Deployment Attributes on Vagrant.

Topics

- Obtaining Attribute Values Directly
- Obtaining Attribute Values with Chef Search

Obtaining Attribute Values Directly

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This approach works only for Linux stacks.

Mocking the Stack Configuration and Deployment Attributes on Vagrant shows how to obtain stack configuration and deployment data by using node syntax to directly reference particular attributes. This is sometimes the best approach. However, many attributes are defined in collections or lists whose contents and names can vary from stack to stack and over time for a particular stack. For example, the `deploy` attribute contains a list of app attributes, which are named with the app's short name. This list, including the app attribute names, typically varies from stack to stack and even from deployment to deployment.

It is often more useful, and sometimes even necessary, to obtain the required data by enumerating the attributes in a list or collection. For example, suppose that you want to know the public IP addresses of your stack's instances. That information is in the `['opsworks']['layers']` attribute, which is set to a hash table that contains one element for each of the stack's layers, named with the layer's shortname. Each layer element is set to a hash table containing the layer's attributes, one of which is `['instances']`. That element in turn is set to yet another hash table containing an attribute for each of the layer's instances, named with the instance's shortname. Each instance attribute is set to still another hash table that contains the instance attributes, including `['ip']`, which represents the public IP address. If you are having trouble visualizing this, the following procedure includes an example in JSON format.

This example shows how to obtain data from the stack configuration and deployment JSON for a stack's layers.

To set up the cookbook

1. Create a directory within `opsworks_cookbooks` named `listip` and navigate to it.
2. Initialize and configure Test Kitchen, as described in Example 1: Installing Packages.
3. Add two directories to `listip`: `recipes` and `environments`.
4. Create an edited JSON version of the MyStack configuration and deployment attributes that contains the relevant attributes. It should look something like the following.

```
{
  "opsworks": {
```

```
    "layers": {
      "php-app": {
        "name": "PHP App Server",
        "id": "efd36017-ec42-4423-b655-53e4d3710652",
        "instances": {
          "php-app1": {
            "ip": "192.0.2.0"
          }
        }
      },
      "db-master": {
        "name": "MySQL",
        "id": "2d8e0b9a-0d29-43b7-8476-a9b2591a7251",
        "instances": {
          "db-master1": {
            "ip": "192.0.2.5"
          }
        }
      },
      "lb": {
        "name": "HAProxy",
        "id": "d5c4dda9-2888-4b22-b1ea-6d44c7841193",
        "instances": {
          "lb1": {
            "ip": "192.0.2.10"
          }
        }
      }
    }
  }
}
```

5. Create an environment file named `test.json`, paste the example JSON into `default_attributes`, and save the file to the cookbook's `environments` folder. The file should look something the following (for brevity, most of the example JSON is represented by an ellipsis).

```
{
  "default_attributes" : {
    "opsworks": {
      "layers": {
        ...
```

```
      }
    }
  },
  "chef_type" : "environment",
  "json_class" : "Chef::Environment"
}
```

6. Replace the text in `.kitchen.yml` with the following.

```
---
driver:
  name: vagrant

provisioner:
  name: chef_zero
  environments_path: ./environment

platforms:
  - name: ubuntu-12.04

suites:
  - name: listip
    provisioner:
      client_rb:
        environment: test
    run_list:
      - recipe[listip::default]
    attributes:
```

After the cookbook is set up, you can use the following recipe to log the layer IDs.

```
node['opsworks']['layers'].each do |layer, layerdata|
  log "#{layerdata['name']} : #{layerdata['id']}"
end
```

The recipe enumerates the layers in `['opsworks']['layers']` and logs each layer's name and ID.

To run the layer ID logging recipe

1. Create a file named `default.rb` with the example recipe and save it to the `recipes` directory.
2. Run `kitchen converge`.

The relevant part of the output should look something like the following.

```
Recipe: listip::default
  * log[PHP App Server : efd36017-ec42-4423-b655-53e4d3710652] action
 write[2014-07-17T22:56:19+00:00] INFO: Processing log[PHP App Server : efd36017-
ec42-4423-b655-53e4d3710652] action write (listip::default line 4)
[2014-07-17T22:56:19+00:00] INFO: PHP App Server : efd36017-ec42-4423-b655-53e4d3710652



  * log[MySQL : 2d8e0b9a-0d29-43b7-8476-a9b2591a7251] action
 write[2014-07-17T22:56:19+00:00] INFO: Processing log[MySQL : 2d8e0b9a-0d29-43b7-8476-
a9b2591a7251] action write (listip::default line 4)
[2014-07-17T22:56:19+00:00] INFO: MySQL : 2d8e0b9a-0d29-43b7-8476-a9b2591a7251


  * log[HAProxy : d5c4dda9-2888-4b22-b1ea-6d44c7841193] action
 write[2014-07-17T22:56:19+00:00] INFO: Processing log[HAProxy : d5c4dda9-2888-4b22-
b1ea-6d44c7841193] action write (listip::default line 4)
[2014-07-17T22:56:19+00:00] INFO: HAProxy : d5c4dda9-2888-4b22-b1ea-6d44c7841193
```

To list the instances' IP addresses, you will need a nested loop like the following.

```
node['opsworks']['layers'].each do |layer, layerdata|
  log "#{layerdata['name']} : #{layerdata['id']}"
  layerdata['instances'].each do |instance, instancedata|
    log "Public IP: #{instancedata['ip']}"
  end
end
```

The inner loop iterates over each layer's instances and logs the IP addresses.

To run the instance IP logging recipe

1. Replace the code in `default.rb` with the example recipe.
2. Run `kitchen converge` to execute the recipe.

The relevant part of the output should look something like the following.

```
  * log[PHP App Server : efd36017-ec42-4423-b655-53e4d3710652] action
 write[2014-07-17T23:09:34+00:00] INFO: Processing log[PHP App Server : efd36017-
ec42-4423-b655-53e4d3710652] action write (listip::default line 2)
[2014-07-17T23:09:34+00:00] INFO: PHP App Server : efd36017-ec42-4423-b655-53e4d3710652



  * log[Public IP: 192.0.2.0] action write[2014-07-17T23:09:34+00:00] INFO: Processing
 log[Public IP: 192.0.2.0] action write (listip::default line 4)
[2014-07-17T23:09:34+00:00] INFO: Public IP: 192.0.2.0


  * log[MySQL : 2d8e0b9a-0d29-43b7-8476-a9b2591a7251] action
 write[2014-07-17T23:09:34+00:00] INFO: Processing log[MySQL : 2d8e0b9a-0d29-43b7-8476-
a9b2591a7251] action write (listip::default line 2)
[2014-07-17T23:09:34+00:00] INFO: MySQL : 2d8e0b9a-0d29-43b7-8476-a9b2591a7251


  * log[Public IP: 192.0.2.5] action write[2014-07-17T23:09:34+00:00] INFO: Processing
 log[Public IP: 192.0.2.5] action write (listip::default line 4)
[2014-07-17T23:09:34+00:00] INFO: Public IP: 192.0.2.5


  * log[HAProxy : d5c4dda9-2888-4b22-b1ea-6d44c7841193] action
 write[2014-07-17T23:09:34+00:00] INFO: Processing log[HAProxy : d5c4dda9-2888-4b22-
b1ea-6d44c7841193] action write (listip::default line 2)
[2014-07-17T23:09:34+00:00] INFO: HAProxy : d5c4dda9-2888-4b22-b1ea-6d44c7841193


  * log[Public IP: 192.0.2.10] action write[2014-07-17T23:09:34+00:00] INFO: Processing
 log[Public IP: 192.0.2.10] action write (listip::default line 4)
[2014-07-17T23:09:34+00:00] INFO: Public IP: 192.0.2.10
```

When you are finished, run `kitchen destroy`; the next topic uses a new cookbook.

> **Note**
>
> One of the most common reasons for enumerating a stack configuration and deployment JSON collection is to obtain data for a particular deployed app, such as its deployment directory. For an example, see Deploy Recipes.

Obtaining Attribute Values with Chef Search

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This approach is available for Windows stacks and Chef 11.10 Linux stacks.

Obtaining stack configuration and deployment attribute values directly from the node object can be complicated, and can't be used with Windows stacks. An alternative approach is to use Chef search to query for the attributes of interest. If you are familiar with Chef server, you will find that Chef search works a bit differently with OpsWorks Stacks. Because OpsWorks Stacks uses chef-client in local mode, Chef search depends on a local version of Chef server called chef-zero, so that search operates on the data that is stored locally in the instance's node object instead of on a remote server.

As a practical matter, restricting search to locally stored data usually doesn't matter because the node object on an OpsWorks Stacks instance includes the stack configuration and deployment attributes. They contain most if not all of the data that recipes would typically obtain from Chef server and use the same names, so you can usually use search code written for Chef server on OpsWorks Stacks instances without modification. For more information, see Using Chef Search.

The following shows the basic structure of a search query:

```
result = search(:search_index, "key:pattern")
```

- The search index specifies what attributes the query applies to and determines the type of object that is returned.
- The key specifies the attribute name.
- The pattern specifies which values of the attribute that you want to retrieve.

  You can query for specific attribute values or use wild cards to query for a range of values.
- The result is a list of objects that satisfy the query, each of which is a hash table containing multiple related attributes.

  For example, if you use the node search index, the query returns a list of instance objects, one for each instance that satisfies the query. Each object is a hash table that contains a set of attributes that define the instance configuration, such as the hostname and IP address.

For example, the following query uses the node search index, which is a standard Chef index that applies to the stack's instances (or nodes, in Chef terminology). It searches for instances with hostname of `myhost`.

```
result = search(:node, "hostname:myhost")
```

Search returns a list of instance objects whose hostname is `myhost`. If you want the first instance's operating system, for example, it would be represented by `result[0][:os]`. If the query returns multiple objects, you can enumerate them to retrieve the required information.

The details of how to use search in a recipe depend on whether you are using a Linux or Windows stack. The following topics provide examples for both stack types.

Topics

- Using Search on a Linux Stack
- Using Search on a Windows Stack

Using Search on a Linux Stack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This example is based on a Linux stack with a single PHP application server. It uses Chef search to obtain the server's public IP address and puts the address in a file in the `/tmp` directory. It retrieves essentially the same information from the node object as Obtaining Attribute Values Directly , but the code is much simpler and does not depend on the details of the stack configuration and deployment attribute structure.

The following briefly summarizes how to create the stack for this example. For more information, see Create a New Stack.

> **Note**
>
> If you have not run a custom recipe on an OpsWorks Stacks instance before, you should first go through the Running a Recipe on a Linux Instance example.

Create a stack

1. Open the OpsWorks Stacks console and click **Add Stack**.
2. Specify the following settings, accept the defaults for the other settings, and click **Add Stack**.

   - **Name** – SearchJSON
   - **Default SSH key** – An Amazon EC2 key pair

   If you need to create an Amazon EC2 key pair, see Amazon EC2 Key Pairs. Note that the key pair must belong to the same AWS region as the instance. The example uses the US West (Oregon) region.
3. Click **Add a layer** and add a PHP App Server layer to the stack with default settings.

4. Add a 24/7 instance with default settings to the layer and start it.

To set up the cookbook

1. Create a directory within opsworks_cookbooks named searchjson and navigate to it.
2. Create a metadata.rb file with the following content and save it to opstest.

```
name "searchjson"
version "0.1.0"
```

3. Create a recipes directory within searchjson.
4. Create a default.rb file with the following recipe and save it to the recipes directory.

```
phpserver = search(:node, "layers:php-app").first
Chef::Log.info("**********The public IP address is: '#{phpserver[:ip]}'**********")

file "/tmp/ip_addresses" do
  content "#{phpserver[:ip]}"
  mode 0644
  action :create
end
```

Linux stacks support only the node search index. The recipe uses this index to obtain a list of instances in the php-app layer. Because the layer is known to have only one instance, the recipe simply assigns the first one to phpserver. If the layer has multiple instances, you can enumerate them to retrieve the required information. Each list item is a hash table containing a set of instance attributes. The ip attribute is set to the instance's public IP address, so you can represent that address in the subsequent recipe code as phpserver[:ip].

After adding a message to the Chef log, the recipe then uses a file resource to create a file named ip_addresses. The content attribute is set to a string representation of phpserver[:ip]. When Chef creates ip_addresses, it adds that string to the file.

5. Create a .zip archive of opsworks_cookbooks, Upload the archive to an Amazon S3 bucket, make the archive public, and record the archive's URL. For more information on cookbook repositories, see Cookbook Repositories.

Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

You can now install the cookbook and run the recipe.

To run the recipe

1. Edit the stack to enable custom cookbooks, and specify the following settings.

   - **Repository type** – **Http Archive**
   - **Repository URL** – The cookbook archive URL that you recorded earlier

   Use the default values for the other settings and click **Save** to update the stack configuration.
2. Edit the custom layer configuration and assign `searchjson::default` to the layer's Setup event. OpsWorks Stacks will run the recipe after the instance boots or if you explicitly trigger the Setup event.
3. Run the Update Custom Cookbooks stack command, which installs the current version of your custom cookbook repository on the stack's instances. If an earlier version of the repository is present, this command overwrites it.
4. Execute the recipe by running the **Setup** stack command, which triggers a Setup event on the instance and runs `searchjson::default`. Leave the **Running command setup page** open.

After the recipe has run successfully, you can verify it.

To verify searchjson

1. The first step is to examine the Chef log for the most recent Setup event. On the **Running command setup page**, click **show** in the php-app1 instance's **Log** column to display the log. Scroll down to find your log message near the middle, which will look something like the following.

```
...
[2014-09-05T17:08:41+00:00] WARN: Previous
 bash[logdir_existence_and_restart_apache2]: ...
```

```
[2014-09-05T17:08:41+00:00] WARN: Current
 bash[logdir_existence_and_restart_apache2]: ...
[2014-09-05T17:08:41+00:00] INFO: **********The public IP address is:
 '192.0.2.0'**********
[2014-09-05T17:08:41+00:00] INFO: Processing directory[/etc/sysctl.d] action create
 (opsworks_initial_setup::sysctl line 1)
...
```

2. Use SSH to log in to the instance and list the contents of `/tmp`, which should include a file named `ip_addresses` that contains the IP address.

Using Search on a Windows Stack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks provides two options to use search on Windows stacks.

- The node search index, which can be used to query a set of standard Chef attributes.

  If you have existing recipes with search code that uses node, they will usually work on OpsWorks Stacks stacks without modification.
- An additional set of search indexes that can be used to query sets of OpsWorks Stacks-specific attributes, and some standard attributes.

  These indexes are discussed in Using OpsWorks Stacks-Specific Search Indexes on Windows Stacks.

We recommend using node for retrieving standard information, such as hostnames or IP addresses. That approach will keep your recipes consistent with standard Chef practice. Use the OpsWorks Stacks search indexes to retrieve information that is specific to OpsWorks Stacks.

Topics

- Using the node Search Index on Windows Stacks
- Using OpsWorks Stacks-Specific Search Indexes on Windows Stacks

Using the node Search Index on Windows Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This example assumes that you have already done the Running a Recipe on a Windows Instance example. If not, you should do that example first. In particular, it describes how to enable RDP access to your instances.

This example is based on a Windows stack with a single custom layer and one instance. It uses Chef search with the node search index to obtain the server's public IP address and puts the address in a file in the `C:\tmp` directory. The following briefly summarizes how to create the stack for this example. For more information, see Create a New Stack.

Create a stack

1. Open the OpsWorks Stacks console and choose **Add Stack**.
2. Specify the following settings, accept the defaults for the other settings, and choose **Add Stack**.

   - **Name** – NodeSearch
   - **Region** – US West (Oregon)

     This example will work in any region, but we recommend using US West (Oregon) for tutorials.

- **Default operating system** – Microsoft Windows Server 2012 R2

3. Choose **Add a layer** and add a custom layer to the stack with the following settings.

   - **Name** – IPTest
   - **Short name** – iptest

4. Add a 24/7 t2.micro instance with default settings to the IPTest layer and start it. It will be named iptest1.

   OpsWorks Stacks automatically assigns `AWS-OpsWorks-RDP-Server` to this instance, which allows authorized users to log in to the instance.

5. Choose **Permissions** and then **Edit**, and select **SSH/RDP** and **sudo/admin**. Regular users need this authorization in addition to the `AWS-OpsWorks-RDP-Server` security group to log in to the instance.

> **Note**
>
> You also can log in as Administrator, but it requires a different procedure. For more information, see Logging In with RDP.

To set up the cookbook

1. Create a directory named `nodesearch` and navigate to it.
2. Create a `metadata.rb` file with the following content and save it to `opstest`.

```
name "nodesearch"
version "0.1.0"
```

3. Create a `recipes` directory within `nodesearch`.
4. Create a `default.rb` file with the following recipe and save it to the `recipes` directory.

```
directory 'C:\tmp' do
  rights :full_control, 'Everyone'
  recursive true
  action :create
end
```

```
windowsserver = search(:node, "hostname:iptest*").first
Chef::Log.info("**********The public IP address is:
 '#{windowsserver[:ipaddress]}'**********")

file 'C:\tmp\addresses.txt' do
  content "#{windowsserver[:ipaddress]}"
  rights :full_control, 'Everyone'
  action :create
end
```

The recipe does the following:

1. Uses a directory resource to create a `C:\tmp` directory for the file.

   For more information on this resource, see Example 3: Creating Directories.

2. Uses Chef search with the `node` search index to obtain a list of nodes (instances) with a hostname that starts with `iptest`.

   If you use the default theme, which creates hostnames by appending integers to the layer's short name, this query will return every instance in the IPTest layer. For this example, the layer is known to have only one instance, so the recipe simply assigns the first one to `windowsserver`. For multiple instances, you can get the complete list and then enumerate them.

3. Adds a message with the IP address to the Chef log for this run.

   The `windowsserver` object is a hash table whose `ipaddress` attribute is set to the instance's public IP address, so you can represent that address in the subsequent recipe code as `windowsserver[:ipaddress]`. The recipe inserts the corresponding string into the message and adds it to the Chef log.

4. Uses the `file` resource to create a file with the IP address named `C:\tmp\addresses.txt`.

   The resource's `content` attribute specifies content to be added to the file, which is the public IP address in this case.

5. Create a `.zip` archive of `nodesearch`, Upload the archive to an S3 bucket, make the archive public, and record the archive's URL.

Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

You can now install the cookbook and run the recipe.

To install the cookbook and run the recipe

1. Edit the stack to enable custom cookbooks and specify the following settings.

   - **Repository type** – **S3 Archive**
   - **Repository URL** – The cookbook archive URL that you recorded earlier

   Accept the default values for the other settings, and choose **Save** to update the stack configuration.
2. Run the Update Custom Cookbooks stack command, which installs the current version of your custom cookbooks on the stack's instances, including online instances. If an earlier version of your cookbooks is present, this command overwrites it.
3. After Update Custom Cookbooks has finished, execute the recipe by running the **Execute Recipes** stack command with **Recipes to execute** set to **`nodesearch::default`**. This command initiates a Chef run, with a run list that consists of your recipe. Leave the execute_recipes page open.

After the recipe has run successfully, you can verify it.

To verify nodesearch

1. Examine the Chef log for the most recent execute_recipes event. On the **Running command execute_recipes page**, choose **show** in the iptest1 instance's **Log** column to display the log. Scroll down to find your log message near the bottom, which will look something like the following.

```
...
[2015-05-13T18:55:47+00:00] INFO: Storing updated cookbooks/nodesearch/recipes/
default.rb in the cache.
```

```
[2015-05-13T18:55:47+00:00] INFO: Storing updated cookbooks/nodesearch/metadata.rb
 in the cache.
[2015-05-13T18:55:47+00:00] INFO: **********The public IP address is:
 '192.0.0.1'**********
[2015-05-13T18:55:47+00:00] INFO: Processing directory[C:\tmp] action create
 (nodesearch::default line 1)
[2015-05-13T18:55:47+00:00] INFO: Processing file[C:\tmp\addresses.txt] action
 create (nodesearch::default line 10)
...
```

2. Use RDP to log in to the instance and examine the contents of `C:\tmp\addresses.txt`.

Using OpsWorks Stacks-Specific Search Indexes on Windows Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This example assumes that you have already done the Running a Recipe on a Windows Instance example. If not, you should do that example first. In particular, it describes how to enable RDP access to your instances.

OpsWorks Stacks provides the following search indexes in addition to node:

- `aws_opsworks_stack` – The stack configuration.
- `aws_opsworks_layer` – The stack's layer configurations.
- `aws_opsworks_instance` – The stack's instance configurations.
- `aws_opsworks_app` – The stack's app configurations.
- `aws_opsworks_user` – The stack's user configurations.
- `aws_opsworks_rds_db_instance` – Connection information for registered RDS instances.

These indexes include some standard Chef attributes, but are primarily intended for retrieving OpsWorks Stacks-specific attributes. For example `aws_opsworks_instance` includes a `status` attribute that provides the instance's status, such as `online`.

> **Note**
>
> The recommended practice is to use `node` when possible to keep your recipes consistent with standard Chef usage. For an example, see Using the node Search Index on Windows Stacks.

This example shows how to use the OpsWorks Stacks indexes to retrieve the value of an OpsWorks Stacks-specific attribute. It is based on a simple Windows stack with a custom layer that has one instance. It uses Chef search to obtain the instance's OpsWorks Stacks ID and puts the results in the Chef log.

The following briefly summarizes how to create a stack for this example. For more information, see Create a New Stack.

Create a stack

1. Open the OpsWorks Stacks console and choose **+ Stack**. Specify the following settings, accept the defaults for the other settings, and choose **Add Stack**.

   - **Name** – IDSearch
   - **Region** – US West (Oregon)

     This example will work in any region, but we recommend using US West (Oregon) for tutorials.
   - **Default operating system** – Microsoft Windows Server 2012 R2
2. Choose **Add a layer** and add a custom layer to the stack with the following settings.

   - **Name** – IDCheck
   - **Short name** – idcheck
3. Add a 24/7 t2.micro instance with default settings to the IDCheck layer and start it. It will be named iptest1.

OpsWorks Stacks automatically assigns `AWS-OpsWorks-RDP-Server` to this instance. Enabling RDP Access explains how to add an inbound rule to this security group that allows authorized users to log in to the instance.

4. Choose **Permissions** and then **Edit**, and choose **SSH/RDP** and **sudo/admin**. Regular users need this authorization in addition to the `AWS-OpsWorks-RDP-Server` security group to log in to the instance.

> **Note**
>
> You can also log in as Administrator, but it requires a different procedure. For more information, see Logging In with RDP.

To set up the cookbook

1. Create a directory named `idcheck` and navigate to it.
2. Create a `metadata.rb` file with the following content and save it to `opstest`.

```
name "idcheck"
version "0.1.0"
```

3. Create a `recipes` directory within `idcheck` and add a `default.rb` file to the directory that contains the following recipe.

```
windowsserver = search(:aws_opsworks_instance, "hostname:idcheck*").first
Chef::Log.info("**********The public IP address is:
 '#{windowsserver[:instance_id]}'**********")
```

The recipe uses Chef search with an `aws_opsworks_instance` search index to obtain the instance attributes of each instance in the stack with a hostname that starts with `idcheck`. If you use the default theme, which creates hostnames by appending integers to the layer's short name, this query will return every instance in the IDCheck layer. For this example, the layer is known to have only one instance, so the recipe simply assigns the first one to `windowsserver`. For multiple instances, you can get the complete list and then enumerate them.

The recipe takes advantage of the fact that there is only one instance in the stack with this hostname, so the first result is the correct one. If your stack has multiple instances, searching on other attributes might return more than one result. For a list of instance attributes, see Instance Data Bag (aws_opsworks_instance).

The instance attributes are basically a hash table, and the instance's OpsWorks Stacks ID is assigned to the `instance_id` attribute, so you can refer to the ID as `windowsserver[:instance_id]`. The recipe inserts the corresponding string into the message and adds it to the Chef log.

4. Create a `.zip` archive of the `ipaddress` cookbook, Upload the archive to an Amazon S3 bucket, and record the archive's URL. For more information on cookbook repositories, see Cookbook Repositories.

   Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

You can now install the cookbook and run the recipe.

To install the cookbook and run the recipe

1. Edit the stack to enable custom cookbooks and specify the following settings.

   - **Repository type** – **S3 Archive**
   - **Repository URL** – The cookbook archive URL that you recorded earlier

   Accept the default values for the other settings, and choose **Save** to update the stack configuration.

2. Run the Update Custom Cookbooks stack command, which installs the current version of your custom cookbooks on the stack's instances, including online instances. If an earlier version of your cookbooks is present, this command overwrites it.
3. After Update Custom Cookbooks is finished, execute the recipe by running the **Execute Recipes** stack command with **Recipes to execute** set to **`idcheck::default`**. This command initiates a Chef run, with a run list that consists of your recipe. Leave the execute_recipes page open.

After the recipe has run successfully, you can verify it by examining the Chef log for the most recent execute_recipes event. On the **Running command execute_recipes page**, choose **show** in the iptest1 instance's **Log** column to display the log. Scroll down to find your log message near the bottom, which will look something like the following.

```
...
[2015-05-13T20:03:47+00:00] INFO: Storing updated cookbooks/nodesearch/recipes/
default.rb in the cache.
[2015-05-13T20:03:47+00:00] INFO: Storing updated cookbooks/nodesearch/metadata.rb in
 the cache.
[2015-05-13T20:03:47+00:00] INFO: **********The instance ID is: 'i-8703b570'**********
[2015-05-13T20:03:47+00:00] INFO: Chef Run complete in 0.312518 seconds
...
```

Using an External Cookbook on a Linux Instance: Berkshelf

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

Berkshelf is available only for Chef 11.10 Linux stacks.

Before you start implementing a cookbook, check out Chef Community Cookbooks, which contains cookbooks that have been created by members of the Chef community for a wide variety of purposes. Many of these cookbooks can be used with OpsWorks Stacks without modification, so you might able to take advantage of them for some of your tasks instead of implementing all the code yourself.

To use an external cookbook on an instance, you need a way to install it and manage any dependencies. The preferred approach is to implement a cookbook that supports a dependency

manager named Berkshelf. Berkshelf works on Amazon EC2 instances, including OpsWorks Stacks instances, but it is also designed to work with Test Kitchen and Vagrant. However, the usage on Vagrant is a bit different than with OpsWorks Stacks, so this topic includes examples for both platforms. For more information on how to use Berkshelf, see Berkshelf.

Topics

- Using Berkshelf with Test Kitchen and Vagrant
- Using Berkshelf with OpsWorks Stacks

Using Berkshelf with Test Kitchen and Vagrant

This example shows how to use Berkshelf to install the getting-started community cookbook and execute its recipe, which installs a brief text file in your home directory on the instance.

To install Berkshelf and initialize a cookbook

1. On your workstation, install the Berkshelf gem, as follows.

   ```
   gem install berkshelf
   ```

   Depending on your workstation, this command might require `sudo`, or you can also use a Ruby environment manager such as RVM. To verify that Berkshelf was successfully installed, run `berks --version`.

2. The cookbook for this topic is named external_cookbook. You can use Berkshelf to create an initialized cookbook instead of the manual approach that the previous topics have taken. To do so, navigate to the `opsworks_cookbooks` directory and run the following command.

   ```
   berks cookbook external_cookbook
   ```

   The command creates the `external_cookbook` directory and several standard Chef and Test Kitchen subdirectories, including `recipes` and `test`. The command also creates default versions of a number of standard files, including the following:

   - `metadata.rb`
   - Configuration files for Vagrant, Test Kitchen, and Berkshelf
   - An empty `default.rb` recipe in the `recipes` directory

> **Note**
>
> You don't need to run `kitchen init`; the `berks cookbook` command handles those tasks.

3. Run `kitchen converge`. The newly created cookbook doesn't do anything interesting at this point, but it does converge.

> **Note**
>
> You can also use `berks init` to initialize an existing cookbook to use Berkshelf.

To use Berkshelf to manage a cookbook's external dependencies, the cookbook's root directory must contain a `Berksfile`, which is a configuration file that specifies how Berkshelf should manage dependencies. When you used `berks cookbook` to create the `external_cookbook` cookbook, it created a `Berksfile` with the following contents.

```
source "https://supermarket.chef.io"
metadata
```

This file has the following declarations:

- `source` – The URL of a cookbook source.

  A Berksfile can have any number of `source` declarations, each of which specifies a default source for dependent cookbooks. If you do not explicitly specify a cookbook's source, Berkshelf looks in the default repositories for a cookbook with the same name. The default Berksfile includes a single `source` attribute which specifies the community cookbook repository. That repository contains the getting-started cookbook, so you can leave the line unchanged.
- `metadata` – Directs Berkshelf to include cookbook dependencies that are declared in the cookbook's `metadata.rb` file.

  You can also declare a dependent cookbook in the Berksfile by including a `cookbook` attribute, as discussed later.

There are two ways to declare a cookbook dependency:

- By including a `cookbook` declaration in the Berksfile.

  This is the approach used by OpsWorks Stacks. For example to specify the getting-started cookbook used in this example, include `cookbook "getting-started"` in the Berksfile. Berkshelf will then look in the default repositories for a cookbook with that name. You can also use `cookbook` to explicitly specify a cookbook source, and even a particular version. For more information, see Berkshelf.
- By including a `metadata` declaration in the Berksfile and declaring the dependency in `metadata.rb`.

  This declaration directs Berkshelf to include cookbook dependencies that are declared in `metadata.rb`. For example, to declare a getting-started dependency, add a `depends 'getting-started'` declaration to the cookbook's `metadata.rb` file.

This example uses the first approach, for consistency with OpsWorks Stacks.

To install the getting-started cookbook

1. Edit the default Berksfile to replace the `metadata` declaration with a `cookbook` declaration for `getting-started`. The contents should look like the following.

   ```
   source "https://supermarket.chef.io"

   cookbook 'getting-started'
   ```
2. Run `berks install`, which downloads the getting-started cookbook from the community cookbook repository to your workstation's Berkshelf directory, which is typically `~/.berkshelf`. This directory is often simply called *the Berkshelf*. Look in the Berkshelf's `cookbooks` directory, and you should see the directory for the getting-started cookbook, which will be named something like `getting-started-0.4.0`.
3. Replace `external_cookbook::default` in the `.kitchen.yml` run list with `getting-started::default`. This example doesn't run any recipes from external_cookbook; it's basically just a way to use the getting-started cookbook. The `.kitchen.yml` file should now look like the following.

```
---
driver:
  name: vagrant

provisioner:
  name: chef_solo

platforms:
  - name: ubuntu-12.04

suites:
  - name: default
    run_list:
      - recipe[getting-started::default]
    attributes:
```

4. Run `kitchen converge` and then use `kitchen login` to log in to the instance. The login directory should contain a file named `chef-getting-started.txt` with something like the following:

```
Welcome to Chef!

This is Chef version 11.12.8.
Running on ubuntu.
Version 12.04.
```

   Test Kitchen installs cookbooks in the instance's `/tmp/kitchen/cookbooks` directory. If you list the contents of that directory, you will see two cookbooks: external_cookbook and getting-started.

5. Run `kitchen destroy` to shut down the instance. The next example uses an OpsWorks Stacks instance.

Using Berkshelf with OpsWorks Stacks

OpsWorks Stacks optionally supports Berkshelf for Chef 11.10 stacks. To use Berkshelf with your stack, you must do the following.

- Enable Berkshelf for the stack.

OpsWorks Stacks then handles the details of installing Berkshelf on the stack's instances.

- Add a Berksfile to your cookbook repository's root directory.

  The Berksfile should contain `source` and `cookbook` declarations for all dependent cookbooks.

When OpsWorks Stacks installs your custom cookbook repository on an instance, it uses Berkshelf to install the dependent cookbooks that are declared in the repository's Berksfile. For more information, see Using Berkshelf.

This example shows how to use Berkshelf to install the getting-started community cookbook on an OpsWorks Stacks instance. It also installs a version of the createfile custom cookbook, which creates a file in a specified directory. For more information on how createfile works, see Installing a File from a Cookbook.

> **Note**
>
> If this is the first time you have installed a custom cookbook on an OpsWorks Stacks stack, you should first go through the Running a Recipe on a Linux Instance example.

Start by creating a stack, as summarized in the following. For more information, see Create a New Stack.

Create a stack

1. Open the OpsWorks Stacks console and click **Add Stack**.
2. Specify the following settings, accept the defaults for the other settings, and click **Add Stack**.

   - **Name** – BerksTest
   - **Default SSH key** – An Amazon EC2 key pair

   If you need to create an Amazon EC2 key pair, see Amazon EC2 Key Pairs. Note that the key pair must belong to the same AWS region as the instance. The example uses the default US West (Oregon) region.
3. Click **Add a layer** and add a custom layer to the stack with the following settings.

   - **Name** – BerksTest

- **Short name** – berkstest

  You could actually use any layer type for this example. However, the example doesn't require any of the packages that are installed by the other layers, so a custom layer is the simplest approach.

4. Add a 24/7 instance to the BerksTest layer with default settings, but don't start it yet.

With OpsWorks Stacks, cookbooks must be in a remote repository with a standard directory structure. You then provide the download information to OpsWorks Stacks, which automatically downloads the repository to each of the stack's instances on startup. For simplicity, the repository for this example is a public Amazon S3 archive, but OpsWorks Stacks also supports HTTP archives, Git repositories, and Subversion repositories. For more information, see Cookbook Repositories.

Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

To create the cookbook repository

1. In your `opsworks_cookbooks` directory, create a directory named `berkstest_cookbooks`. If you prefer, you can create this directory anywhere that you find convenient, because you will upload it to a repository.
2. Add a file named Berksfile to `berkstest_cookbooks` with the following contents.

   ```
   source "https://supermarket.chef.io"

   cookbook 'getting-started'
   ```

   This file declares the getting-started cookbook dependency, and directs Berkshelf to download it from the community cookbook site.
3. Add a `createfile` directory to `berkstest_cookbooks` that contains the following.

   - A `metadata.rb` file with the following contents.

     ```
     name "createfile"
     ```

```
version "0.1.0"
```

- A `files/default` directory that contains an `example_data.json` file with the following content.

```
{
  "my_name" : "myname",
  "your_name" : "yourname",
  "a_number" : 42,
  "a_boolean" : true
}
```

  The file's name and content are arbitrary. The recipe simply copies the file to the specified location.

- A `recipes` directory that contains a `default.rb` file with the following recipe code.

```
directory "/srv/www/shared" do
  mode 0755
  owner 'root'
  group 'root'
  recursive true
  action :create
end

cookbook_file "/srv/www/shared/example_data.json" do
  source "example_data.json"
  mode 0644
  action :create_if_missing
end
```

  This recipe creates `/srv/www/shared` and copies `example_data.json` to that directory from the cookbook's `files` directory.

4. Create a `.zip` archive of `berkstest_cookbooks`, Upload the archive to an Amazon S3 bucket, make the archive public, and record the archive's URL.

You can now install the cookbooks and run the recipe.

To install the cookbooks and run the recipes

1. Edit the stack to enable custom cookbooks, and specify the following settings.

   - **Repository type** – **Http Archive**
   - **Repository URL** – The cookbook archive URL that you recorded earlier
   - **Manage Berkshelf** – **Yes**

   The first two settings provide OpsWorks Stacks with the information it needs to download the cookbook repository to your instances. The last setting enables Berkshelf support, which downloads the getting-started cookbook to the instance. Accept the default values for the other settings and click **Save** to update the stack configuration.
2. Edit the BerksTest layer to add the following recipes to the layer's Setup lifecycle event.

   - `getting-started::default`
   - `createfile::default`
3. Start the instance. The Setup event occurs after the instance finishes booting. OpsWorks Stacks then installs the cookbook repository, uses Berkshelf to download the getting-started cookbook, and runs the layer's setup and deploy recipes, including `getting-started::default` and `createfile::default`.
4. After the instance is online, use SSH to log in. You should see the following

   - `/srv/www/shared` should contain `example_data.json`.
   - `/root` should contain `chef-getting-started.txt`.

     OpsWorks Stacks runs recipes as root, so getting-started installs the file in the `/root` directory rather than your home directory.

Using the SDK for Ruby: Downloading Files from Amazon S3

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

There are some tasks, such as interacting with AWS services, that cannot be handled with Chef resources. For example, it is sometimes preferable to store files remotely and have a recipe download them to the instance. You can use the remote_file resource to download files from remote servers. However, if you want to store your files in an Amazon S3 bucket, `remote_file` can download those files only if the ACL allows the operation.

Recipes can use the AWS SDK for Ruby to access most AWS services. This topic shows how to use the SDK for Ruby to download a file from an S3 bucket.

Note

For more information about how to use the AWS SDK for Ruby to handle encryption and decryption, see AWS::S3::S3Object. Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

Topics

- Using the SDK for Ruby on a Vagrant Instance
- Using the SDK for Ruby on an OpsWorks Stacks Linux Instance
- Using the SDK for Ruby on an OpsWorks Stacks Windows Instance

Using the SDK for Ruby on a Vagrant Instance

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This topic describes how a recipe running on a Vagrant instance can use the AWS SDK for Ruby to download a file from Amazon S3. Before starting, you must first have a set of AWS credentials—an access key and a secret access key—that allow the recipe to access Amazon S3.

> **Important**
>
> We strongly recommend that you do not use root account credentials for this purpose. Instead, create a user with an appropriate policy and provide those credentials to the recipe.
>
> Be careful not to put credentials—even IAM user credentials—in a publicly accessible location, such as by uploading a file containing the credentials to a public GitHub or Bitbucket repository. Doing so exposes your credentials and could compromise your account's security.
>
> Recipes running on an EC2Amazon EC2 instance can use an even better approach, an IAM role, as described in Using the SDK for Ruby on an OpsWorks Stacks Linux Instance.
>
> Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

If you don't already have an appropriate user, you can create one as follows. For more information, see What is IAM.

> **Warning**
>
> IAM users have long-term credentials, which presents a security risk. To help mitigate this risk, we recommend that you provide these users with only the permissions they require to perform the task and that you remove these users when they are no longer needed.

To create an IAM user

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users** and, if necessary, choose **Add users** to create a new administrative user.
3. On the **Set permissions** page, choose **Attach policies directly**.

4. Type **S3** in the **Permissions policies** search box to display the Amazon S3 policies.

   Choose **AmazonS3ReadOnlyAccess**. If you prefer, you can specify a policy that grants broader permissions, such as **AmazonS3FullAccess**, but standard practice is to grant only those permissions that are required. In this case, the recipe will only be downloading a file, so read-only access is sufficient.
5. Choose **Next**.
6. Choose **Create user**
7. Next create access keys for your user. For more information about creating access keys, see Managing access keys for IAM users in the *IAM User Guide*.

You must next provide a file to be downloaded. This example assumes that you will put a file named `myfile.txt` in a newly created S3 bucket named `cookbook_bucket`.

To provide a file for downloading

1. Create a file named `myfile.txt` with the following text and save it in a convenient location on your workstation.

   ```
   This is the file that you just downloaded from Amazon S3.
   ```
2. On the Amazon S3 console, create a bucket named `cookbook_bucket` in the **Standard** region and upload `myfile.txt` to the bucket.

Set the cookbook up as follows.

To set up the cookbook

1. Create a directory within `opsworks_cookbooks` named `s3bucket` and navigate to it.
2. Initialize and configure Test Kitchen, as described in Example 1: Installing Packages.
3. Replace the text in `.kitchen.yml` with the following.

   ```
   ---
   driver:
     name: vagrant

   provisioner:
   ```

```
  name: chef_solo
  environments_path: ./environments

platforms:
  - name: ubuntu-14.04

suites:
  - name: s3bucket
    provisioner:
      solo_rb:
        environment: test
    run_list:
      - recipe[s3bucket::default]
    attributes:
```

4. Add two directories to `s3bucket`: `recipes` and `environments`.
5. Create an environment file named `test.json` with the following `default_attributes` section, replacing the `access_key` and `secret_key` values with the corresponding keys for your user. Save the file to the cookbook's `environments` folder.

```
{
  "default_attributes" : {
    "cookbooks_101" : {
      "access_key": "AKIAIOSFODNN7EXAMPLE",
      "secret_key" : "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
    }
  },
  "chef_type" : "environment",
  "json_class" : "Chef::Environment"
}
```

You have a variety of ways to provide credentials to a recipe running on an instance. The key consideration is limiting the chances of accidentally exposing the keys and compromising your account security. For that reason, using explicit key values in your code is not recommended. The example instead puts the key values in the node object, which allows the recipe to reference them by using node syntax instead of exposing literal values. You must have root privileges to access the node object, which limits the possibility that the keys might be exposed. For more information, see Best Practices for Managing AWS Access Keys.

> **Note**
>
> Notice that the example uses nested attributes, with `cookbooks_101` as the first element. This practice limits the chance of a name collision if there are other `access_key` or `secret_key` attributes in the node object.

The following recipe downloads `myfile.text` from the `cookbook_bucket` bucket.

```
gem_package "aws-sdk ~> 3" do
  action :install
end

ruby_block "download-object" do
  block do
    require 'aws-sdk'

    s3 = Aws::S3::Client.new(
          :access_key_id => "#{node['cookbooks_101']['access_key']}",
          :secret_access_key => "#{node['cookbooks_101']['secret_key']}")

    myfile = s3.bucket['cookbook_bucket'].objects['myfile.txt']
    Dir.chdir("/tmp")
    File.open("myfile.txt", "w") do |f|
      f.write(myfile.read)
      f.close
    end
  end
  action :run
end
```

The first part of the recipe installs the SDK for Ruby, which is a gem package. The gem_package resource installs gems that will be used by recipes or other applications.

> **Note**
>
> Your instance usually has two Ruby instances, which are typically different versions. One is a dedicated instance that is used by the Chef client. The other is used by applications and recipes running on the instance. It's important to understand this distinction when installing gem packages, because there are two resources for installing gems, gem_package

and chef_gem. If applications or recipes use the gem package, install it with gem_package. chef_gem is only for gem packages used by Chef client.

The remainder of the recipe is a ruby_block resource, which contains the Ruby code that downloads the file. You might think that because a recipe is a Ruby application, you could put the code in the recipe directly. However, a Chef run compiles all of that code before executing any resources. If you put the example code directly in the recipe, Ruby will attempt to resolve the `require 'aws-sdk'` statement before it executes the gem_package resource. Because the SDK for Ruby hasn't been installed yet, compilation will fail.

Code in a ruby_block resource isn't compiled until that resource is executed. In this example, the ruby_block resource is executed after the gem_package resource has finished installing the SDK for Ruby, so the code will run successfully.

The code in the ruby_block works as follows.

1. Creates a new Aws::S3 object, which provides the service interface.

   The access and secret keys are specified by referencing the values stored in the node object.
2. Calls the S3 object's bucket.objects association, which returns an Aws::S3::Object object named myfile that represents myfile.txt.
3. Uses Dir.chdir to set the working directory to /tmp.
4. Opens a file named myfile.txt, writes the contents of myfile to the file, and closes the file.

To run the recipe

1. Create a file named default.rb with the example recipe and save it to the recipes directory.
2. Run kitchen converge.
3. Run kitchen login to log in to the instance, and then run ls /tmp. You should see the myfile.txt, along with several Test Kitchen files and directories.

```
vagrant@s3bucket-ubuntu-1204:~$ ls /tmp
install.sh  kitchen  myfile.txt  stderr
```

You can also run `cat /tmp/myfile.txt` to verify that the file's content is correct.

When you are finished, run `kitchen destroy` to terminate the instance.

Using the SDK for Ruby on an OpsWorks Stacks Linux Instance

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This topic describes how to use the SDK for Ruby on an OpsWorks Stacks Linux instance to download a file from an Amazon S3 bucket. OpsWorks Stacks automatically installs the SDK for Ruby on every Linux instance. However, when you create a service's client object, you must provide a suitable set of AWS credentials `AWS::S3.new` or the equivalent for other services.

Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

Using the SDK for Ruby on a Vagrant Instance shows how to mitigate the risk of exposing your credentials by storing the credentials in the node object and referencing the attributes in your recipe code. When you run recipes on an Amazon EC2 instance, you have an even better option, an IAM role.

An IAM role works much like an IAM user. It has an attached policy that grants permissions to use the various AWS services. However, you assign a role to an Amazon EC2 instance rather than to an individual. Applications running on that instance can then acquire the permissions granted by the attached policy. With a role, credentials never appear in your code, even indirectly. This topic describes how you can use an IAM role to run the recipe from Using the SDK for Ruby on a Vagrant Instance on an Amazon EC2 instance.

You could run this recipe with Test Kitchen using the kitchen-ec2 driver, as described in Example 9: Using Amazon EC2 Instances. However, installing the SDK for Ruby on Amazon EC2 instances is

somewhat complicated and not something you need to be concerned with for OpsWorks Stacks. All OpsWorks Stacks Linux instances have the SDK for Ruby installed by default. For simplicity, the example therefore uses an OpsWorks Stacks instance.

The first step is to set up the IAM role. This example takes the simplest approach, which is to use the Amazon EC2 role that OpsWorks Stacks creates when you create your first stack. It is named `aws-opsworks-ec2-role`. However, OpsWorks Stacks does not attach a policy to that role, so by default it grants no permissions.

You must attach the `AmazonS3ReadOnlyAccess` policy to the `aws-opsworks-ec2-role` role to grants appropriate permissions. For more information about how to attach a policy to a role, see Adding IAM identity permissions (console) in the *IAM User Guide*.

You specify the role when you create or update a stack. Set up a stack with a custom layer, as described in Running a Recipe on a Linux Instance, with one addition. On the **Add Stack** page, confirm that **Default IAM instance profile** is set to **aws-opsworks-ec2-role**. OpsWorks Stacks will then assign that role to all of the stack's instances.

The procedure for setting up the cookbook is similar to the one used by Running a Recipe on a Linux Instance. The following is a brief summary; you should refer to that example for details.

To set up the cookbook

1. Create a directory named `s3bucket_ops` and navigate to it.
2. Create a `metadata.rb` file with the following content and save it to `s3bucket_ops`.

```
name "s3bucket_ops"
version "0.1.0"
```

3. Create a `recipes` directory within `s3bucket_ops`.
4. Create a `default.rb` file with the following recipe and save it to the `recipes` directory.

```
Chef::Log.info("******Downloading a file from Amazon S3.******")

ruby_block "download-object" do
  block do
    require 'aws-sdk'
```

```
    s3 = AWS::S3.new

    myfile = s3.buckets['cookbook_bucket'].objects['myfile.txt']
    Dir.chdir("/tmp")
    File.open("myfile.txt", "w") do |f|
      f.syswrite(myfile.read)
      f.close
    end
  end
  action :run
end
```

5. Create a `.zip` archive of `s3bucket_ops` and upload the archive to an Amazon S3 bucket. For simplicity, make the archive public, then record the archive's URL for later use. You can also store your cookbooks in a private Amazon S3 archive, or several other repository types. For more information, see Cookbook Repositories.

This recipe is similar the one used by the previous example, with the following exceptions.

- Because OpsWorks Stacks has already installed the SDK for Ruby, the `chef_gem` resource has been deleted.
- The recipe does not pass any credentials to `AWS::S3.new`.

  Credentials are automatically assigned to the application based on the instance's role.
- The recipe uses `Chef::Log.info` to add a message to the Chef log.

Create a stack for this example as follows. You can also use an existing Windows stack. Just update the cookbooks, as described later.

To create a stack

1. Open the OpsWorks Stacks console and click **Add Stack**.
2. Specify the following settings, accept the defaults for the other settings, and click **Add Stack**.
   - **Name** – RubySDK
   - **Default SSH key** – An Amazon EC2 key pair

If you need to create an Amazon EC2 key pair, see Amazon EC2 Key Pairs. Note that the key pair must belong to the same AWS region as the instance. The example uses the default US West (Oregon) region.

3. Click **Add a layer** and add a custom layer to the stack with the following settings.

   - **Name** – S3Download
   - **Short name** – s3download

   Any layer type will actually work for Linux stacks, but the example doesn't require any of the packages that are installed by the other layer types, so a custom layer is the simplest approach.

4. Add a 24/7 instance with default settings to the layer and start it.

You can now install and run the recipe

To run the recipe

1. Edit the stack to enable custom cookbooks, and specify the following settings.

   - **Repository type** – **Http Archive**
   - **Repository URL** – The cookbook's archive URL that you recorded earlier.

   Use the default values for the other settings and click **Save** to update the stack configuration.

2. Run the Update Custom Cookbooks stack command, which installs the current version of your custom cookbooks on the stack's instances. If an earlier version of your cookbooks is present, this command overwrites it.
3. Execute the recipe by running the **Execute Recipes** stack command with **Recipes to execute** set to **`s3bucket_ops::default`**. This command initiates a Chef run, with a run list that consists of `s3bucket_ops::default`.

   > **Note**
   >
   > You typically have OpsWorks Stacks run your recipes automatically by assigning them to the appropriate lifecycle event. You can run such recipes by manually triggering

the event. You can use a stack command to trigger Setup and Configure events, and a deploy command to trigger Deploy and Undeploy events.

After the recipe runs successfully, you can verify it.

To verify s3bucket_ops

1. The first step is to examine the Chef log. Your stack should have one instance named opstest1. On the **Instances** page, click **show** in the instance's **Log** column to display the Chef log. Scroll down and to find your log message near the bottom.

```
...
[2014-07-31T17:01:45+00:00] INFO: Storing updated cookbooks/opsworks_cleanup/
attributes/customize.rb in the cache.
[2014-07-31T17:01:45+00:00] INFO: Storing updated cookbooks/opsworks_cleanup/
metadata.rb in the cache.
[2014-07-31T17:01:46+00:00] INFO: ******Downloading a file from Amazon S3.******
[2014-07-31T17:01:46+00:00] INFO: Processing template[/etc/hosts] action create
 (opsworks_stack_state_sync::hosts line 3)
...
```

2. Use SSH to log in to the instance and list the contents of `/tmp`.

Using the SDK for Ruby on an OpsWorks Stacks Windows Instance

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This example assumes that you have already done the Running a Recipe on a Windows Instance example. If not, you should do that example first. In particular, it describes how to enable RDP access to your instances.
> Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

This topic describes how to use the AWS SDK for Ruby on an OpsWorks Stacks Windows instance to download a file from an S3 bucket.

If a Ruby application needs to access an AWS resource, you must provide it with a set of AWS credentials with the appropriate permissions. For recipes, your best option for providing AWS credentials is to use an AWS Identity and Access Management (IAM) role. An IAM role works much like an IAM user it has an attached policy that grants permissions to use the various AWS services. However, you assign a role to an Amazon Elastic Compute Cloud (Amazon EC2) instance instead of to an individual. Applications running on that instance can then acquire the permissions granted by the attached policy. With a role, credentials never appear in your code, even indirectly.

The first step is to set up the IAM role. This example takes the simplest approach, which is to use the Amazon EC2 role that OpsWorks Stacks creates when you create your first stack. It is named `aws-opsworks-ec2-role`. However, OpsWorks Stacks does not attach a policy to that role, so by default it grants no permissions.

You must attach the `AmazonS3ReadOnlyAccess` policy to the `aws-opsworks-ec2-role` role to grants appropriate permissions. For more information about how to attach a policy to a role, see Adding IAM identity permissions (console) in the *IAM User Guide*.

You specify the role when you create or update a stack. Set up a stack with a custom layer, as described in Running a Recipe on a Windows Instance, with one addition. On the **Add Stack** page, confirm that **Default IAM instance profile** is set to **aws-opsworks-ec2-role**. OpsWorks Stacks will then assign that role to all of the stack's instances.

The procedure for setting up the cookbook is similar to the one used by Running a Recipe on a Linux Instance. The following is a brief summary; refer to that example for details.

To set up the cookbook

1. Create a directory named `s3bucket_ops` and navigate to it.
2. Create a `metadata.rb` file with the following content and save it to `s3bucket_ops`.

```
name "s3download"
version "0.1.0"
```

3. Create a `recipes` directory within `s3download`.
4. Create a `default.rb` file with the following recipe, and save it to the `recipes` directory. Replace *`windows-cookbooks`* with the name of the S3 bucket that you will use to store the file to be downloaded.

```
Chef::Log.info("******Downloading an object from S3******")

chef_gem "aws-sdk-s3" do
  compile_time false
  action :install
end

ruby_block "download-object" do
  block do
    require 'aws-sdk-s3'

    Aws.use_bundled_cert!

    s3_client = Aws::S3::Client.new(region:'us-west-2')

    s3_client.get_object(bucket: 'windows-cookbooks',
                         key: 'myfile.txt',
                         response_target: '/chef/myfile.txt')
  end
  action :run
end
```

5. Create a `.zip` archive of `s3download` and upload the file to an S3 bucket. Make the file public and record the URL for later use.
6. Create a text file named `myfile.txt` and upload it to an S3 bucket. This is the file that your recipe will download, so you can use any convenient bucket.

The recipe performs the following tasks.

1: Install the SDK for Ruby v2.

The example uses the SDK for Ruby to download the object. However, OpsWorks Stacks does not install this SDK on Windows instances, so the first part of the recipe uses a `chef_gem` resource to handle that task. You use this resource to install gems for use by Chef, which includes recipes.

2: Download the file.

The third part of the recipe uses a `ruby_block` resource to run SDK for Ruby v2 code to download `myfile.txt` from an S3 bucket named *`windows-cookbooks`* to the instance's `/chef` directory. Change *`windows-cookbooks`* to the name of the bucket that contains `myfile.txt`.

> **Note**
>
> A recipe is a Ruby application, so you can put Ruby code in the body of the recipe; it doesn't have to be in a `ruby_block` resource. However, Chef executes the Ruby code in the recipe's body first, followed by each resource, in order. For this example, if you put the download code in the recipe's body, it will fail because it depends on the SDK for Ruby, and the `chef_gem` resource that installs the SDK hasn't yet executed. The code in the `ruby_block` resource executes when the resource executes, and that happens after the `chef_gem` resource has installed the SDK for Ruby.

Create a stack for this example as follows. You can also use an existing Windows stack. Just update the cookbooks, as described later.

Create a stack

1. Open the OpsWorks Stacks console and choose **Add Stack**. Specify the following settings, accept the defaults for the other settings, and choose **Add Stack**.
   - **Name** – S3Download
   - **Region** – US West (Oregon)

This example will work in any region, but we recommend using US West (Oregon) for tutorials.

- **Default operating system** – Microsoft Windows Server 2012 R2

2. Choose **Add a layer** and add a custom layer to the stack with the following settings.

   - **Name** – S3Download
   - **Short name** – s3download

3. Add a 24/7 instance with default settings to the S3Download layer and start it.

You can now install and run the recipe

To run the recipe

1. Edit the stack to enable custom cookbooks, and specify the following settings.

   - **Repository type** – **S3 Archive**.
   - **Repository URL** – The cookbook's archive URL that you recorded earlier.

   Accept the default values for the other settings and choose **Save** to update the stack configuration.

2. Run the Update Custom Cookbooks stack command, which installs the latest version of your custom cookbook on the stack's online instances. If an earlier version of your cookbooks is present, this command overwrites it.

3. Execute the recipe by running the **Execute Recipes** stack command with **Recipes to execute** set to **`s3download::default`**. This command initiates a Chef run, with a run list that consists of `s3download::default`.

> **Note**
>
> You typically have OpsWorks Stacks run your recipes automatically by assigning them to the appropriate lifecycle event. You also can run such recipes by manually triggering the event. You can use a stack command to trigger Setup and Configure events, and a deploy command to trigger Deploy and Undeploy events.

After the recipe runs successfully, you can verify it.

To verify s3download

1. The first step is to examine the Chef log. Your stack should have one instance named s3download1. On the **Instances** page, choose **show** in the instance's **Log** column to display the Chef log. Scroll down to find your log message near the bottom.

```
...
[2015-05-01T21:11:04+00:00] INFO: Loading cookbooks [s3download@0.0.0]
[2015-05-01T21:11:04+00:00] INFO: Storing updated cookbooks/s3download/recipes/
default.rb in the cache.
[2015-05-01T21:11:04+00:00] INFO: ******Downloading an object from S3******
[2015-05-01T21:11:04+00:00] INFO: Processing chef_gem[aws-sdk] action install
 (s3download::default line 3)
[2015-05-01T21:11:05+00:00] INFO: Processing ruby_block[download-object] action run
 (s3download::default line 8)
...
```

2. Use RDP to log in to the instance and examine the contents of `c:\chef`.

Installing Windows Software

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> These examples assume that you have already done the Running a Recipe on a Windows Instance example. If not, you should do that example first. In particular, it describes how to enable RDP access to your instances.

Windows instances start with Windows Server 2012 R2 Standard, so you typically need to install some software. The details depend on the type of software.

- Windows features are optional system components, including the .NET frameworks and Internet Information Services (IIS), which you can download to your instance.
- Third-party software typically comes in an installer package, such as an MSI file, which you must download to the instance and then run.

  Some Microsoft software also comes in an installer package.

This section describes how to implement cookbooks to install Windows features and packages. It also introduces the Chef windows cookbook, which contains resources and helper functions that simplify implementing recipes for Windows instances.

Topics

- Installing a Windows Feature: IIS
- Installing a Package on a Windows Instance

Installing a Windows Feature: IIS

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Windows features are a set of optional system components, including the .NET frameworks and Internet Information Services (IIS). This topic describes how to implement a cookbook to install a commonly used feature, Internet Information Services (IIS).

> **Note**
>
> Installing a Package shows how to install software that comes in an installer package, such as an MSI file, which you must download to the instance and run. IIS cookbooks

Running a Recipe on a Windows Instance shows how to use a `powershell_script` resource to install a Windows feature. This example shows an alternative approach: use the Chef Windows cookbook's `windows_feature` resource. This cookbook contains a set of resources that use Deployment Image Servicing and Management to perform a variety of tasks on Windows, including feature installation.

> **Note**
>
> Chef also has an IIS cookbook, which you can use to manage IIS. For more information, see IIS cookbook.

To set up the cookbook

1. Go to the windows cookbook GitHub repository and download the `windows` cookbook.

   This example assumes that you will download the `windows` repository as a .zip file, but you can also clone the repository if you prefer.
2. Go to the chef_handler cookbook GitHub repository and download the `chef-handler` cookbook.

   The `windows` cookbook depends on `chef_handler`; you won't be using it directly. This example assumes that you will download the `chef_handler` repository as a .zip file, but you can also clone the repository if you prefer.
3. Extract the `windows` and `chef_handler` cookbooks to directories in your cookbooks directory named `windows` and `chef_handler`, respectively.
4. Create a directory in your cookbooks directory named `install-iis` and navigate to it.
5. Add a `metadata.rb` file to `install-iis` with the following content.

   ```
   name "install-iis"
   version "0.1.0"

   depends "windows"
   ```

   The `depends` directive allows you to use the `windows` cookbook resources in your recipes.
6. Add a `recipes` directory to `install-iis` and add a file named `default.rb` to that directory that contains the following recipe code.

```
%w{ IIS-WebServerRole IIS-WebServer }.each do |feature|
  windows_feature feature do
    action :install
  end
end

service 'w3svc' do
  action [:start, :enable]
end
```

The recipe uses the `windows` cookbook's `windows_feature` resource to install the following:

1. The IIS Web Server role.
2. The IIS Web Server.

The recipe then uses a `service` resource to start and enable the IIS service (W3SVC).

> **Note**
>
> For a complete list of available Windows features, use RDP to log in to the instance, open a command prompt window, and run the following command. Note that the list is quite long.
>
> ```
> dism /online /Get-Features
> ```

7. Create a `.zip` archive that contains the `install-iis`, `chef_handler`, and `windows` cookbooks and upload the archive to an S3 bucket. Make the archive public and record the URL for later use. This example assumes that the archive is named `install-iis.zip`. For more information, see Cookbook Repositories.

   Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

Create a stack for this example as follows. You also can use an existing Windows stack. Just update the cookbooks, as described later.

Create a stack

1. Open the OpsWorks Stacks console and choose **Add Stack**. Specify the following settings, accept the defaults for the other settings, and choose **Add Stack**.

   - **Name** – InstallIIS
   - **Region** – US West (Oregon)

     This example will work in any region, but we recommend using US West (Oregon) for tutorials.
   - **Default operating system** – Microsoft Windows Server 2012 R2
2. Choose **Add a layer** and add a custom layer to the stack with the following settings.

   - **Name** – IIS
   - **Short name** – iis
3. Add a 24/7 instance with default settings to the IIS layer and start it.

You can now install the cookbook and run the recipe

To install the cookbook and run the recipe

1. Edit the stack to enable custom cookbooks, and specify the following settings.

   - **Repository type** – **S3 Archive**
   - **Repository URL** – The cookbook archive's URL that you recorded earlier.

   Accept the default values for the other settings and choose **Save** to update the stack configuration.
2. Run the **Update Custom Cookbooks stack** command, which installs the latest version of your custom cookbooks on the stack's online instances. If an earlier version of your cookbooks is present, this command overwrites it.
3. Execute the recipe by running the **Execute Recipes** stack command with **Recipes to execute** set to **`install-iis::default`**. This command initiates a Chef run, which runs the specified recipes.

> **Note**
>
> This example uses **Execute Recipes** for convenience, but you typically have OpsWorks Stacks run your recipes automatically by assigning them to the appropriate lifecycle event. You can run such recipes by manually triggering the event. You can use a stack command to trigger Setup and Configure events, and a deploy command to trigger Deploy and Undeploy events.

4. To verify the installation, use RDP to connect to the instance and open Windows Explorer. The file system should now have a `C:\inetpub` directory. If you check the list of services in the Administrative Tools Control Panel application, IIS should be near the bottom. However, it will be named World Wide Web Publishing Service, not IIS.

Installing a Package on a Windows Instance

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This example assumes that you have already done the Running a Recipe on a Windows Instance example. If not, you should do that example first. In particular, it describes how to enable RDP access to your instances.

If your software comes in an installer package, such as an MSI, you must download the file to the instance and then run it. This example shows how to implement a cookbook to install an MSI package, the Python runtime, including how to define associated environment variables. For more information on how to install Windows features such as IIS, see Installing a Windows Feature: IIS.

To set up the cookbook

1. Create a directory named `installpython` and navigate to it.
2. Add a `metadata.rb` file to `installpython` with the following content.

```
name "installpython"
version "0.1.0"
```

3. Add `recipes` and `files` directories to `installpython` and add a `default` directory to files.
4. Download a Python package from Python Releases for Windows to the cookbook's `files\default` directory. This example installs the Windows x86-64 version of Python 3.5.0a3, which uses an MSI installer named `python-3.4.3.amd64.msi`.
5. Add a file named `default.rb` to the `recipes` directory with the following recipe code.

```
directory 'C:\tmp' do
  rights :full_control, 'Everyone'
  recursive true
  action :create
end

cookbook_file 'C:\tmp\python-3.4.3.amd64.msi' do
  source "python-3.4.3.amd64.msi"
  rights :full_control, 'Everyone'
  action :create
end

windows_package 'python' do
  source 'C:\tmp\python-3.4.3.amd64.msi'
  action :install
end

env "PATH" do
  value 'c:\python34'
  delim ";"
  action :modify
end
```

The recipe does the following:

1. Uses a directory resource to create a `C:\tmp` directory.

   For more information on this resource, see Example 3: Creating Directories.

2. Uses a cookbook_file resource to copy the installer from the cookbook's `files\default` directory to `C:\tmp`.

   For more information on this resource, see Installing a File from a Cookbook.

3. Uses a windows_package resource to run the MSI installer, which installs Python to `c:\python34`.

   The installer creates the required directories and installs the files, but does not modify the system's PATH environment variable.

4. Uses an env resource to add `c:\python34` to the system path.

   You use the env resource to define environment variables. In this case, the recipe allows you to easily run Python scripts from the command line by adding `c:\python34` to the path.

   - The resource name specifies the environment variable's name, PATH for this example.
   - The `value` attribute specifies the variable's value, `c:\\python34` for this example (you need to escape the \ character).
   - The `:modify` action prepends the specified value to the variable's current value.
   - The `delim` attribute specifies a delimiter that separates the new value from the existing value, which is `;` for this example.

6. Create a `.zip` archive of `installpython`, upload the archive to an S3 bucket, and make it public. Record the archive's URL for later use. For more information, see Cookbook Repositories.

   Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

Create a stack for this example as follows. You also can use an existing Windows stack. Just update the cookbooks, as described later.

Create a stack

1. Open the OpsWorks Stacks console and choose **Add Stack**. Specify the following settings, accept the defaults for the other settings, and choose **Add Stack**.

   - **Name** – InstallPython
   - **Region** – US West (Oregon)

     This example will work in any region, but we recommend using US West (Oregon) for tutorials.
   - **Default operating system** – Microsoft Windows Server 2012 R2
2. Choose **Add a layer** and add a custom layer to the stack with the following settings.

   - **Name** – Python
   - **Short name** – python
3. Add a 24/7 instance with default settings to the Python layer and start it.

After the instance is online, you can install the cookbook and run the recipe

To install the cookbook and run the recipe

1. Edit the stack to enable custom cookbooks, and specify the following settings.

   - **Repository type** – **S3 Archive**.
   - **Repository URL** – The cookbook's archive URL that you recorded earlier.

   Accept the default values for the other settings and choose **Save** to update the stack configuration.
2. Run the **Update Custom Cookbooks** stack command, which installs the latest version of your custom cookbooks on the stack's online instances. If an earlier version of your cookbook is present, this command overwrites it.
3. Execute the recipe by running the **Execute Recipes** stack command with **Recipes to execute** set to **`installpython::default`**. This command initiates a Chef run, with a run list that consists of `installpython::default`.

> **Note**
>
> This example uses **Execute Recipes** for convenience, but you typically have OpsWorks Stacks run your recipes automatically by assigning them to the appropriate lifecycle event. You can run such recipes by manually triggering the event. You can use a stack command to trigger Setup and Configure events, and a deploy command to trigger Deploy and Undeploy events.

4. To verify the installation, use RDP to connect to the instance and open Windows Explorer.

   - The file system should now have a `C:\Python34` directory.
   - If you run `path` from the command line, it should look something like: `PATH=c:\python34;C:\Windows\system32;...`
   - If you run `python --version` from the command line, it should return `Python 3.4.3`.

Overriding Built-In Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This topic applies only to Linux stacks. You cannot override built-in attributes on Windows stacks.

OpsWorks Stacks installs a set of built-in cookbooks on each instance. Many of the built-in cookbooks support the built-in layers, and their attribute files define a variety of default system and application settings, such as the Apache server configuration settings. By putting these

settings in attribute files, you can customize many configuration settings by overriding the corresponding built-in attribute in either of the following ways:

- Define the attribute in custom JSON.

  This approach has the advantage of being simple and flexible. However, you must enter custom JSON manually, so there is no robust way to manage the attribute definitions.
- Implement a custom cookbook and define the attribute in a `customize.rb` attribute file.

  This approach is less flexible than using custom JSON, but is more robust because you can put custom cookbooks under source control.

This topic describes how to use a custom cookbook attribute file to override built-in attributes, using the Apache server as an example. For more information on how to override attributes with custom JSON, see Using Custom JSON. For a general discussion of how to override attributes, see Overriding Attributes.

> **Note**
>
> Overriding attributes is the preferred way to customize configuration settings, but settings are not always represented by attributes. In that case, you can often customize the configuration file by overriding the template that the built-in recipes use to create the configuration file. For an example, see Overriding Built-In Templates.

The built-in attributes typically represent values in the template files that Setup recipes use to create configuration files. For example, one of the `apache2` Setup recipes, `default.rb`, uses the `apache2.conf.erb` template to create the Apache server's main configuration file, `httpd.conf` (Amazon Linux) or `apache2.conf` (Ubuntu). The following is an excerpt from the template file:

```
...
#
# MaxKeepAliveRequests: The maximum number of requests to allow
# during a persistent connection. Set to 0 to allow an unlimited amount.
# We recommend you leave this number high, for maximum performance.
#
MaxKeepAliveRequests <%= node[:apache][:keepaliverequests] %>
#
```

```
# KeepAliveTimeout: Number of seconds to wait for the next request from the
# same client on the same connection.
#
KeepAliveTimeout <%= node[:apache][:keepalivetimeout] %>
##
## Server-Pool Size Regulation (MPM specific)
##

...
```

The `KeepAliveTimeout` setting in this example is the value of the `[:apache][:keepalivetimeout]` attribute. This attribute's default value is defined in the `apache2` cookbook's `apache.rb` attribute file, as shown in the following excerpt:

```
...
# General settings
default[:apache][:listen_ports] = [ '80','443' ]
default[:apache][:contact] = 'ops@example.com'
default[:apache][:log_level] = 'info'
default[:apache][:timeout] = 120
default[:apache][:keepalive] = 'Off'
default[:apache][:keepaliverequests] = 100
default[:apache][:keepalivetimeout] = 3
...
```

Note

For more information about commonly used built-in attributes, see Built-in Cookbook Attributes.

To support overriding built-in attributes, all built-in cookbooks contain a `customize.rb` attribute file, which is incorporated into all modules through an `include_attribute` directive. The built-in cookbooks' `customize.rb` files contain no attribute definitions and have no effect on the built-in attributes. To override the built-in attributes, you create a custom cookbook with the same name as the built-in cookbook and put your custom attribute definitions in an attribute file that is also named `customize.rb`. That file takes precedence over the built-in version, and is included in any related modules. If you define any built-in attributes in your `customize.rb`, they override the corresponding built-in attributes.

This example shows how to override the built-in `[:apache][:keepalivetimeout]` attribute to set its value to 5 instead of 3. You can use a similar approach for any built-in attribute. However, be careful which attributes you override. For example, overriding attributes in the `opsworks` namespace might cause problems for some built-in recipes.

> **Important**
>
> Do not override built-in attributes by modifying a copy of the built-in attributes file itself. For example, you *could* put a copy of `apache.rb` in your custom cookbook's `apache2/attributes` folder and modify some of its settings. However, this file takes precedence over the built-in version, and the built-in recipes will now use your version of `apache.rb`. If OpsWorks Stacks later modifies the built-in `apache.rb` file, recipes will not get the new values unless you manually update your version. By using `customize.rb`, you override only the specified attributes; the built-in recipes continue to automatically get up-to-date values for every attribute that you have not overridden.

To start, create a custom cookbook.

To create the cookbook

1. Within your `opsworks_cookbooks` directory, create a cookbook directory named `apache2` and navigate to it.

   To override built-in attributes, the custom cookbook must have the same name as the built-in cookbook, `apache2` for this example.
2. In the `apache2` directory, create an `attributes` directory.
3. Add a file named `customize.rb` to the `attributes` directory and use it to define the built-in cookbook attributes that you want to override. For this example, the file should contain the following:

   ```
   normal[:apache][:keepalivetimeout] = 5
   ```

   > **Important**
   >
   > To override a built-in attribute, a custom attribute must be a `normal` type or higher and have exactly the same node name as the corresponding built-in attribute. The

`normal` type ensures that the custom attribute takes precedence over the built-in attributes, which are all `default` type. For more information, see Attribute Precedence.

4. Create a `.zip` archive of `opsworks_cookbooks` named `opsworks_cookbooks.zip` and upload the archive to an Amazon Simple Storage Service (Amazon S3) bucket. For simplicity, make the file public. Record the URL for later use. You can also store your cookbooks in a private Amazon S3 archive or in other repository types. For more information, see Cookbook Repositories.

   Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

To use the custom attribute, create a stack and install the cookbook.

To use the custom attribute

1. Open the OpsWorks Stacks console, and then choose **Add Stack**.
2. Specify the following standard settings.

   - **Name** – ApacheConfig
   - **Region** – US West (Oregon)

     You can put your stack in any region, but we recommend US West (Oregon) for tutorials.
   - **Default SSH key** – An EC2 key pair

     If you need to create an EC2 key pair, see Amazon EC2 Key Pairs. Note that the key pair must belong to the same AWS region as the stack.

   Choose **Advanced>>**, set **Use custom Chef cookbooks** to **Yes**, and then specify the following settings.

   - **Repository type** – **Http Archive**
   - **Repository URL** – The cookbook archive's URL that you recorded earlier

   Accept the defaults for the other settings, and then choose **Add Stack** to create the stack.

> **Note**
>
> This example uses the default operating system, Amazon Linux. You can use Ubuntu, if you prefer. The only difference is that on Ubuntu systems, the built-in Setup recipe produces a configuration file with the same settings named `apache2.conf` and puts it in the `/etc/apache2` directory.

3. Choose **Add a layer**, and then add a Java App Server layer with default settings to the stack.
4. Add a 24/7 instance with default settings to the layer, and then start the instance.

   A t2.micro instance is sufficient for this example.
5. After the instance is online, connect to it with SSH. The `httpd.conf` file is in the `/etc/httpd/conf` directory. If you examine the file, you should see your custom `KeepAliveTimeout` setting. The remainder of the settings will have the default values from the built-in `apache.rb` file. The relevant part of `httpd.conf` should look similar to the following:

```
...
#
# KeepAliveTimeout: Number of seconds to wait for the next request from the
# same client on the same connection.
#
KeepAliveTimeout 5
...
```

Overriding Built-In Templates

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This topic applies only to Linux stacks. You cannot override built-in templates on Windows stacks.

The OpsWorks Stacks built-in recipes use templates to create files on instances, primarily configuration files for servers, such as Apache. For example, the `apache2` recipes use the `apache2.conf.erb` template to create the Apache server's primary configuration file, `httpd.conf` (Amazon Linux) or `apache2.conf` (Ubuntu).

Most of the configuration settings in these templates are represented by attributes, so the preferred way to customize a configuration file is by overriding the appropriate built-in attributes. For an example, see Overriding Built-In Attributes. However, if the settings that you want to customize aren't represented by built-in attributes, or aren't in the template at all, you must override the template itself. This topic describes how to override a built-in template to specify a custom Apache configuration setting.

You can provide custom error responses to Apache by adding `ErrorDocument` settings to the `httpd.conf` file. `apache2.conf.erb` contains only some commented-out examples, as shown in the following:

```
...
#
# Customizable error responses come in three flavors:
# 1) plain text 2) local redirects 3) external redirects
#
# Some examples:
#ErrorDocument 500 "The server made a boo boo."
#ErrorDocument 404 /missing.html
#ErrorDocument 404 "/cgi-bin/missing_handler.pl"
#ErrorDocument 402 http://www.example.com/subscription_info.html
...
```

Because these settings are hardcoded comments, you can't specify custom values by overriding attributes; you must override the template itself. However, unlike with attributes, there is no way to override particular parts of a template file. You must create a custom cookbook with the same name as the built-in version, copy the template file to the same subdirectory, and modify the file

as needed. This topic shows how to override `apache2.conf.erb` to provide a custom response to error 500. For a general discussion of overriding templates, see Using Custom Templates.

> **Important**
>
> When you override a built-in template, the built-in recipes use your customized version of the template instead of the built-in version. If OpsWorks Stacks updates the built-in template, the custom template becomes out of sync and might not work correctly. OpsWorks Stacks doesn't make such changes often, and when a template does change, OpsWorks Stacks lists the changes and gives you the option of upgrading to a new version. We recommend that you monitor the OpsWorks Stacks repository for changes, and manually update your custom template as needed. Note that the repository has a separate branch for each supported Chef version, so be sure that you are in the correct branch.

To start, create a custom cookbook.

To create the cookbook

1. In the `opsworks_cookbooks` directory, create a cookbook directory named `apache2`, and then navigate to it. To override built-in templates, the custom cookbook must have the same name as the built-in cookbook, `apache2` for this example.

   > **Note**
   >
   > If you have already completed the Overriding Built-In Attributes walkthrough, you can use the same `apache2` cookbook for this example, and skip Step 2.

2. Create a `metadata.rb` file with the following content, and then save it to the `apache2` directory.

   ```
   name "apache2"
   version "0.1.0"
   ```

3. In `apache2` directory, create a `templates/default` directory..

> **Note**
>
> The `templates/default` directory works for Amazon Linux instances, which use the default `apache2.conf.erb` template. Ubuntu 14.04 instances use an operating system-specific `apache2.conf.erb` template, which is in the `templates/ubuntu-14.04` directory. If you want the customization to apply to Ubuntu 14.04 instances also, you must override that template too.

4. Copy the built-in `apache2.conf.erb` template to your `templates/default` directory. Open the template file, uncomment the `ErrorDocument 500` line, and provide a custom error message, as follows:

```
...
ErrorDocument 500 "A custom error message."
#ErrorDocument 404 /missing.html
...
```

5. Create a `.zip` archive of `opsworks_cookbooks` named `opsworks_cookbooks.zip`, and then upload the file to an Amazon Simple Storage Service (Amazon S3) bucket. For simplicity, make the archive public. Record the archive's URL for later use. You can also store your cookbooks in a private Amazon S3 archive or in other repository types. For more information, see Cookbook Repositories.

   Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

> **Note**
>
> For simplicity, this example adds a hardcoded error message to the template. To change it, you must modify the template and reinstall the cookbook. To give yourself greater flexibility, you can define a default custom attribute for the error string in the custom cookbook's `customize.rb` attribute file and assign the value of that attribute to `ErrorDocument 500`. For example, if you name the attribute `[:apache][:custom][:error500]`, the corresponding line in `apache2.conf.erb` would then look something like the following:

```
...
ErrorDocument 500 <%= node[:apache][:custom][:error500] %>
#ErrorDocument 404 /missing.html
...
```

You can then change the custom error message at any time by overriding `[:apache][:custom][:error500]`. If you use custom JSON to override the attribute, you don't even need to touch the cookbook.

To use the custom template, create a stack and install the cookbook.

To use the custom template

1. Open the OpsWorks Stacks console, and then choose **Add Stack**.
2. Specify the following standard settings:

   - **Name** – ApacheTemplate
   - **Region** – US West (Oregon)
   - **Default SSH key** – An Amazon Elastic Compute Cloud (Amazon EC2) key pair

     If you need to create an Amazon EC2 key pair, see Amazon EC2 Key Pairs. Note that the key pair must belong to the same AWS region as the instance.

   Choose **Advanced>>**, choose **Use custom Chef cookbooks**, to specify the following settings:

   - **Repository type** – **Http Archive**
   - **Repository URL** – The cookbook archive's URL that you recorded earlier

   Accept the default values for the other settings, and then choose **Add Stack** to create the stack.
3. Choose **Add a layer**, and then add a Java App Server layer to the stack with default settings.
4. Add a 24/7 instance with default settings to the layer, and then start the instance.

   A t2.micro instance is sufficient for this example.

5. After the instance is online, connect to it with SSH. The `httpd.conf` file is in the `/etc/httpd/conf` directory. The file should contain your custom `ErrorDocument` setting, which will look something like the following:

```
...
# Some examples:
ErrorDocument 500 "A custom error message."
#ErrorDocument 404 /missing.html
#ErrorDocument 404 "/cgi-bin/missing_handler.pl"
#ErrorDocument 402 http://www.example.com/subscription_info.html
...
```

Load Balancing a Layer

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks provides two load balancing options, Elastic Load Balancing and HAProxy, which are typically used to balance load across an application server layer's instances. This topic describes the benefits and limitations of each to help you decide which option to choose when adding load balancing to a layer. In some cases, the best approach is to use both.

SSL Termination

The built-in HAProxy layer does not handle SSL termination; you must terminate SSL at the servers. The advantage of this approach is that traffic is encrypted until it reaches the servers. However, the servers must handle decryption, which increases server load. In addition, you must put your SSL certificates on the application servers, which are more accessible to users.

With Elastic Load Balancing, you can terminate SSL at the load balancer. This reduces the load on your application servers, but traffic between the load balancer and the server is not

encrypted. Elastic Load Balancing also allows you to terminate SSL at the server, but it is somewhat complicated to set up.

Scaling

If incoming traffic exceeds the capacity of an HAProxy load balancer, you must increase its capacity manually.

Elastic Load Balancing automatically scales to handle incoming traffic. To ensure that an Elastic Load Balancing load balancer has sufficient capacity to handle the expected load when it first comes online, you can pre-warm it.

Load Balancer Failure

If the instance hosting your HAProxy server fails, it could take your entire site offline until you can restart the instance.

Elastic Load Balancing is more failure resistant than HAProxy. For example, it provisions load balancing nodes in each Availability Zone that has registered EC2 instances. If service in one zone is disrupted, the other nodes continue to handle incoming traffic. For more information, see Elastic Load Balancing Concepts.

Idle Timeout

Both load balancers terminate a connection if a server is idle for more than a specified idle timeout value.

- HAProxy – The idle timeout value does not have an upper limit.
- Elastic Load Balancing – The default idle timeout value is 60 seconds, with a maximum of 3600 seconds (60 minutes).

The Elastic Load Balancing idle time limit is sufficient for most purposes. We recommend using HAProxy if you require a longer idle timeout. For example:

- A long-running HTTP connection that is used for push notifications.
- An administrative interface that you use to perform tasks that could take longer than 60 minutes.

URL-based Mapping

You might want to have a load balancer forward an incoming request to a particular server based on the request's URL. For example, suppose you have a group of ten application

servers that supports an online commerce application. Eight of the servers handle the catalog and two handle payments. You want to direct all payment-related HTTP requests to the payment servers, based on the request URL. In this case, you would direct all URLs that include "payment" or "checkout" to one of the payment servers.

With HAProxy, you can use URL-based mapping to direct URLs containing a specified string to particular servers. To use URL-based mapping with OpsWorks Stacks, you must create a custom HAProxy configuration file by overriding the `haproxy-default.erb` template in the `haproxy` built-in cookbook. For more information, see HAProxy Configuration Manual and Using Custom Templates. You cannot use URL-based mapping for HTTPS requests. An HTTPS request is encrypted, so HAProxy has no way to examine the request URL.

Elastic Load Balancing has limited support for URL mapping. For more information, see Listener Configurations for Elastic Load Balancing.

Recommendation: We recommend using Elastic Load Balancing for load balancing unless you have requirements that can be handled only by HAProxy. In that case, the best approach might be combining the two by using Elastic Load Balancing as a front-end load balancer that distributes incoming traffic to a set of HAProxy servers. To do this:

- Set up an HAProxy instance in each of your stack's Availability Zones to distribute requests to the zone's application servers.
- Assign the HAProxy instances to an Elastic Load Balancing load balancer, which then distributes incoming requests to the HAProxy load balancers.

This approach allows you to use HAProxy's URL-based mapping to distribute different types of requests to the appropriate application servers. However, if one of the HAProxy servers goes offline, the site will continue to function because the Elastic Load Balancing load balancer automatically distributes incoming traffic to the healthy HAProxy servers. Note that you must use Elastic Load Balancing as the front-end load balancer; an HAProxy server cannot distribute requests to other HAProxy servers.

Migrating from Chef Server to OpsWorks Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate

their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Because OpsWorks Stacks is based on Chef, migrating from Chef Server to OpsWorks Stacks is relatively straightforward. This topic provides guidelines for modifying Chef Server code to work with OpsWorks Stacks.

Note

We do not recommend migrating to stacks using Chef versions earlier than 11.10, which are based on chef-solo and do not support search or data bags.

Topics

- Mapping Roles to Layers
- Using Data Bags
- Using Chef Search
- Managing Cookbooks and Recipes
- Using Chef Environments

Mapping Roles to Layers

Chef Server uses roles to represent and manage instances with the same purpose and configuration, such as a set of instances that each host a Java application server. An OpsWorks Stacks layer serves essentially the same purpose as a Chef role. A layer is a blueprint for creating a set of Amazon Elastic Compute Cloud (Amazon EC2) instances with the same configuration, installed packages, application deployment procedure, and so on.

OpsWorks Stacks includes a set of built-in layers for several types of application server, an HAProxy load balancer, a MySQL database master, and a Ganglia monitoring master. For example, the built-in Java App Server layer is a blueprint for creating instances that host a Tomcat server.

To migrate to OpsWorks Stacks, you need to associate each role with a layer that provides equivalent functionality. For some roles, you might be able to simply use one of the built-in layers.

Other roles might require varying degrees of customization. Start by examining the functionality of the built-in layers, including the recipes associated with each, to see if one provides at least some of your role's functionality. For more information about the built-in layers, see Layers and OpsWorks Stacks Layer Reference. To examine the built-in recipes, see the OpsWorks Stacks public GitHub repository.

How you proceed depends on how closely you can match a layer to each role, as follows.

A built-in layer supports all of the role's functionality

You can use the built-in layer directly, with minor customizations, if necessary. For example, if a role supports a Tomcat server, the Java App Server layer's recipes might already handle all of the role's tasks, perhaps with some modest customization. For example, you can make the layer's built-in recipes use custom Tomcat or Apache configuration settings by overriding the appropriate attributes or templates.

A built-in layer supports some, but not all, of the role's functionality

You might be able to use a built-in layer by extending the layer. This typically involves implementing custom recipes to support the missing functionality and assigning the recipes to the layer's lifecycle events. For example, suppose that your role installs a Redis server on the same instances that host a Tomcat server. You could extend the Java App Server layer to match the role's functionality by implementing a custom recipe to install Redis on the layer's instances and assigning the recipe to the layer's Setup event.

No built-in layer adequately supports the role's functionality

Implement a custom layer. For example, suppose that your role supports a MongoDB database server, which is not supported by any of the built-in layers. You can provide that support by implementing recipes to install the required packages, configure the server, and so on, and assign the recipes to a custom layer's lifecycle events. Typically, you can use at least some of the role's recipes for this purpose. For more information about how to implement a custom layer, see Creating a Custom Tomcat Server Layer.

Using Data Bags

Chef Server allows you to pass user-defined data to your recipes by using data bags.

- You store the data with your cookbooks, and Chef installs it on each instance.
- You can use encrypted data bags for sensitive data such as passwords.

OpsWorks Stacks supports data bags; recipes can retrieve the data using exactly the same code as with Chef Server. However, the support has the following limitations and differences:

- Data bags are supported only on Chef 11.10 Linux and later stacks.

  Windows stacks and Linux stacks running earlier versions of Chef do not support data bags.
- You do not store data bags in your cookbook repository.

  Instead, you use custom JSON to manage your data bag's data.
- OpsWorks Stacks does not support encrypted data bags.

  If you need to store sensitive data in encrypted form, such as passwords or certificates, we recommend storing it in a private S3 bucket. You can then create a custom recipe that uses the Amazon SDK for Ruby to retrieve the data. For an example, see Using the SDK for Ruby.

For more information, see Using Data Bags.

Using Chef Search

Chef Server stores stack configuration information, such as IP addresses and role configurations, on the server. Recipes use Chef search to retrieve this data. OpsWorks Stacks uses a somewhat different approach. For example, Chef 11.10 Linux stacks are based on Chef client local mode, a Chef client option that runs a lightweight version of Chef Server (often called Chef Zero) locally on the instance. Chef Zero supports search against the data stored in the instance's node object.

Instead of storing stack data on a remote server, OpsWorks Stacks adds a set of stack configuration and deployment attributes to each instance's node object for every lifecycle event. These attributes represent a snapshot of the stack configuration. They use the same syntax as Chef Server and represent most of the data that recipes need to retrieve from the server.

You often don't need to modify your recipes' search-dependent code for OpsWorks Stacks. Because Chef search operates on the node object, which includes the stack configuration and deployment attributes, search queries in OpsWorks Stacks usually work exactly as they do with Chef Server.

The primary exception is caused by the fact that the stack configuration and deployment attributes contain only data that OpsWorks Stacks is aware of when it installs the attributes on the instance. If you create or modify an attribute locally on a particular instance those changes do not propagate back to OpsWorks Stacks and are not incorporated into the stack configuration and deployment

attributes that are installed on the other instances. You can use search to retrieve the attribute value only on that instance. For more information, see Using Chef Search.

For compatibility with Chef Server, OpsWorks Stacks adds a set of `role` attributes to the node object, each of which contains one of the stack's layer attributes. If your recipe uses `roles` as a search key, you don't need to change the search code. The query automatically returns data for the corresponding layer. For example, the following queries both return the `php-app` layer's attributes.

```
phpserver = search(:node, "layers:php-app").first
```

```
phpserver = search(:node, "roles:php-app").first
```

Managing Cookbooks and Recipes

OpsWorks Stacks and Chef Server handle cookbooks and recipes somewhat differently. With Chef Server:

- You provide all of the cookbooks, either by implementing them yourself or by using community cookbooks.
- You store cookbooks on the server.
- You execute recipes manually or on a regular schedule.

With OpsWorks Stacks:

- OpsWorks Stacks provides one or more cookbooks for each of the built-in layers. These cookbooks handle standard tasks, such as installing and configuring a built-in layer's software and deploying apps.

  To handle tasks that aren't performed by the built-in cookbooks, you add custom cookbooks to your stack or use community cookbooks.
- You store OpsWorks Stacks cookbooks in a remote repository, such as an S3 bucket or a Git repository.

  For more information, see Storing Cookbooks.
- You can execute recipes manually, but you typically have OpsWorks Stacks execute recipes for you in response to a set of lifecycle events that occur at key points during an instance's lifecycle.

For more information, see Executing Recipes.

- OpsWorks Stacks supports Berkshelf on Chef 11.10 stacks only. If you use Berkshelf to manage your cookbook dependencies, you cannot use stacks running Chef 11.4 or earlier versions.

  For more information, see Using Berkshelf.

Topics

- Storing Cookbooks
- Executing Recipes

Storing Cookbooks

With Chef Server, you store your cookbooks on the server and deploy them from the server to the instances. With OpsWorks Stacks, you store cookbooks in a repository— an S3 or HTTP archive or a Git or Subversion repository. You specify the information that OpsWorks Stacks needs to download the code from the repository to a stack's instances when you install cookbooks.

To migrate from Chef Server, you must put your cookbooks in one of these repositories. For information on how to structure a cookbook repository, see Cookbook Repositories.

Executing Recipes

In OpsWorks Stacks, each layer has a set of lifecycle events—Setup, Configure, Deploy, Undeploy, and Shutdown—each of which occurs at a key point during an instance's lifecycle. To execute a custom recipe, you typically assign it to the appropriate event on the appropriate layer. When the event occurs, OpsWorks Stacks runs the associated recipes. For example, the Setup event occurs after an instance finishes booting, so you typically assign recipes to this event that perform tasks such as installing and configuring packages and starting services.

You can execute recipes manually by using the Execute Recipes stack command. This command is useful for development and testing, but you also can use it to execute recipes that don't map to a lifecycle event. You can also use the Execute Recipes command to manually trigger Setup and Configure events.

In addition to the OpsWorks Stacks console, you can use the AWS CLI or SDKs to execute recipes. These tools support all of the OpsWorks Stacks API actions, but are simpler to use than the API. Use the create-deployment CLI command to trigger a lifecycle event, which runs all of

the associated recipes. You also can use this command to execute one or more recipes without triggering an event. The equivalent SDK code depends on the particular language, but is generally similar to the CLI command.

The following examples describe two ways to use the `create-deployment` CLI command to automate application deployment.

- Deploy your app on a regular schedule by adding a custom layer with a single instance to your stack.

  Add a custom Setup recipe to the layer that creates a `cron` job on the instance to run the command on a specified schedule. For an example of how to use a recipe to create a `cron` job, see Running Cron Jobs on Linux Instances.
- Add a task to your continuous integration pipeline that uses the `create-deployment` CLI command to deploy the app.

Using Chef Environments

OpsWorks Stacks does not support Chef environments; `node.chef_environment` always returns `_default`.

OpsWorks Stacks Layer Reference

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Every instance that OpsWorks Stacks deploys must be a member of at least one layer, which defines an instance's role in the stack and controls the details of setting up and configuring the instance, installing packages, deploying applications, and so on. For more information about how to use the OpsWorks Stacks to create and manage layers, see Layers.

Each layer description includes a list of the built-in recipes that OpsWorks Stacks runs for each of the layer's lifecycle events. Those recipes are stored at https://github.com/aws/opsworks-

cookbooks. Note that the lists include only those recipes that are run directly by OpsWorks Stacks. Those recipes sometimes run dependent recipes, which are not listed. To see the complete list of recipes for a particular event, including dependent and custom recipes, examine the run list in the appropriate lifecycle event's Chef log.

Topics

- HAProxy Layer Reference
- HAProxy OpsWorks Stacks Layer
- MySQL Layer Reference
- MySQL OpsWorks Layer
- Application Server Layers Reference
- Application Server Layers
- ECS Cluster Layer Reference
- Custom Layer Reference
- Other Layers Reference
- Other Layers

HAProxy Layer Reference

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

This layer is available only for Linux-based stacks.

A HAProxy layer uses HAProxy—a reliable high-performance TCP/HTTP load balancer— to provide high availability load balancing and proxy services for TCP- and HTTP-based applications. It is

particularly useful for websites that must crawl under very high loads while requiring persistence or Layer 7 processing.

HAProxy monitors traffic and displays the statistics and the health of the associated instances on a web page. By default, the URI is http://*DNSName*/haproxy?stats, where *DNSName* is the HAProxy instance's DNS name.

Short name: lb

Compatibility: A HAProxy layer is compatible with the following layers: custom, db-master, and memcached.

Open ports: HAProxy allows public access to ports 22 (SSH), 80 (HTTP), and 443 (HTTPS).

Autoassign Elastic IP addresses: On by default

Default EBS volume: No

Default security group: AWS-OpsWorks-LB-Server

Configuration: To configure a HAProxy layer, you must specify the following:

- Health check URI (default: http://*DNSName*/).
- Statistics URI (default: http://*DNSName*/haproxy?stats).
- Statistics password (optional).
- Health check method (optional). By default, HAProxy uses the HTTP OPTIONS method. You can also specify GET or HEAD.
- Enable statistics (optional)
- Ports. By default, OpsWorks Stacks configures HAProxy to handle both HTTP and HTTPS traffic. You can configure HAProxy to handle only one or the other by overriding the Chef configuration template, `haproxy.cfg.erb`.

Setup recipes:

- opsworks_initial_setup
- ssh_host_keys
- ssh_users

- mysql::client
- dependencies
- ebs
- opsworks_ganglia::client
- haproxy

Configure recipes:

- opsworks_ganglia::configure-client
- ssh_users
- agent_version
- haproxy::configure

Deploy recipes:

- deploy::default
- haproxy::configure

Shutdown recipes:

- opsworks_shutdown::default
- haproxy::stop

Installation:

- OpsWorks Stacks uses the instance's package installer to install HAProxy to its default locations.
- You must set up syslog to direct the log files to a specified location. For more information, see HAProxy.

HAProxy OpsWorks Stacks Layer

> **Note**
>
> This layer is available only for Chef 11 and earlier Linux-based stacks.

The OpsWorks Stacks HAProxy layer is an OpsWorks Stacks layer that provides a blueprint for instances that host an HAProxy server—a reliable high-performance TCP/HTTP load balance. One small instance is usually sufficient to handle all application server traffic.

> **Note**
>
> Stacks are limited to a single region. To distribute your application across multiple regions, you must create a separate stack for each region.

To create a HAProxy layer

1. In the navigation pane, click **Layers**.
2. On the **Layers** page, click **Add a Layer** or **+ Layer**. For **Layer type**, select **HAProxy**.

The layer has the following configuration settings, all of which are optional.

HAProxy statistics

Whether the layer collects and displays statistics. The default value is **Yes**.

Statistics URL

The statistics page's URL path. The complete URL is http://*DNSName**StatisticsPath*, where *DNSName* is the associated instance's DNS name. The default *StatisticsPath* value is /haproxy?stats, which corresponds to something like: http://ec2-54-245-151-7.us-west-2.compute.amazonaws.com/haproxy?stats.

Statistics user name

The statistics page's user name, which you must provide to view the statistics page. The default value is "opsworks".

Statistics password

A statistics page password, which you must provide to view the statistics page. The default value is a randomly generated string.

Health check URL

The health check URL suffix. HAProxy uses this URL to periodically call an HTTP method on each application server instance to determine whether the instance is functioning. If the health

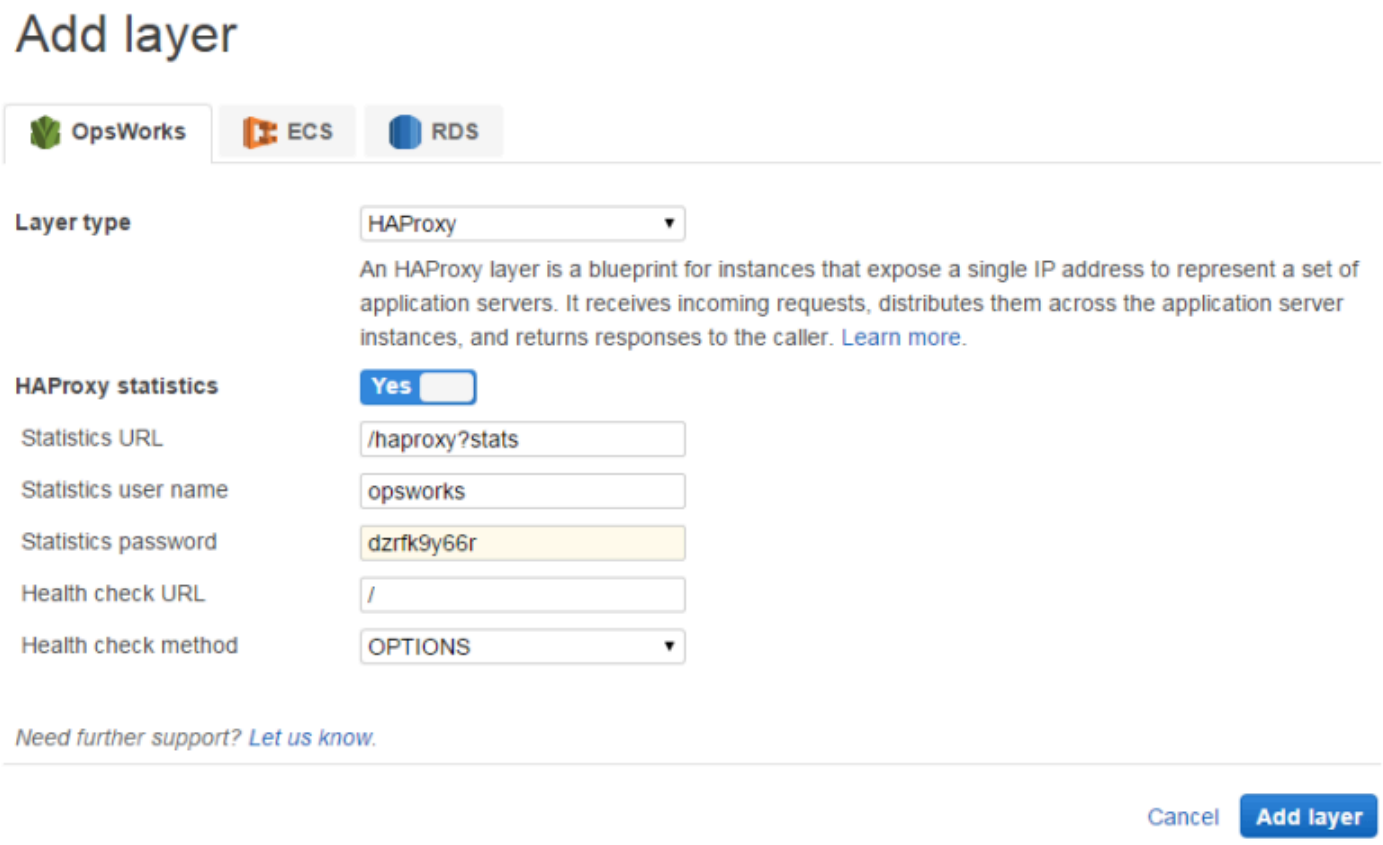
check fails, HAProxy stops routing traffic to the instance until it is restarted, either manually or through auto healing. The default value for the URL suffix is "/", which corresponds to the server instance's home page: http://*DNSName*/.

Health check method

An HTTP method to be used to check whether instances are functioning. The default value is **OPTIONS** and you can also specify **GET** or **HEAD**. For more information, see httpchk.

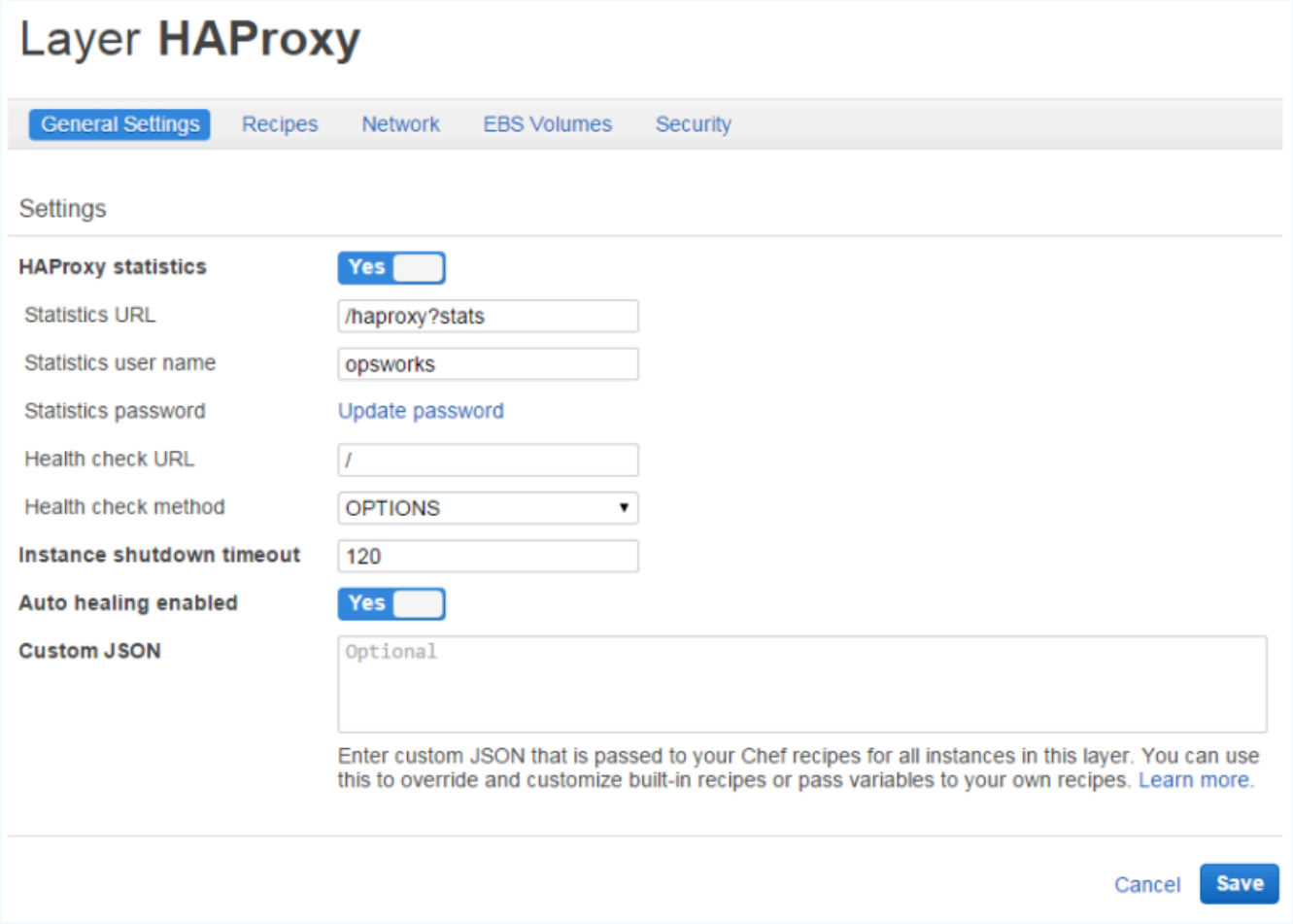
Custom security groups

This setting appears if you chose to not automatically associate a built-in OpsWorks Stacks security group with your layers. You must specify which security group to associate with the layer. Make sure that the group has the correct settings to allow traffic between layers. For more information, see Create a New Stack.



Note

Record the password for later use; OpsWorks Stacks does not allow you to view the password after you create the layer. However, you can update the password by going to the layer's **Edit** page and clicking **Update password** on the **General Settings** tab.



How the HAProxy Layer Works

By default, HAProxy does the following:

- Listens for requests on the HTTP and HTTPS ports.

  You can configure HAProxy to listen on only the HTTP or HTTPS port by overriding the Chef configuration template, `haproxy.cfg.erb`.
- Routes incoming traffic to instances that are members of any application server layer.

By default, OpsWorks Stacks configures HAProxy to distribute traffic to instances that are members of any application server layer. You could, for example, have a stack with both Rails App Server and PHP App Server layers, and an HAProxy master distributes traffic to the instances in both layers. You can configure the default routing by using a custom recipe.

- Routes traffic across multiple Availability Zones.

  If one Availability Zone goes down, the load balancer routes incoming traffic to instances in other zones so your application continues to run without interruption. For this reason, a recommended practice is to distribute your application servers across multiple Availability Zones.

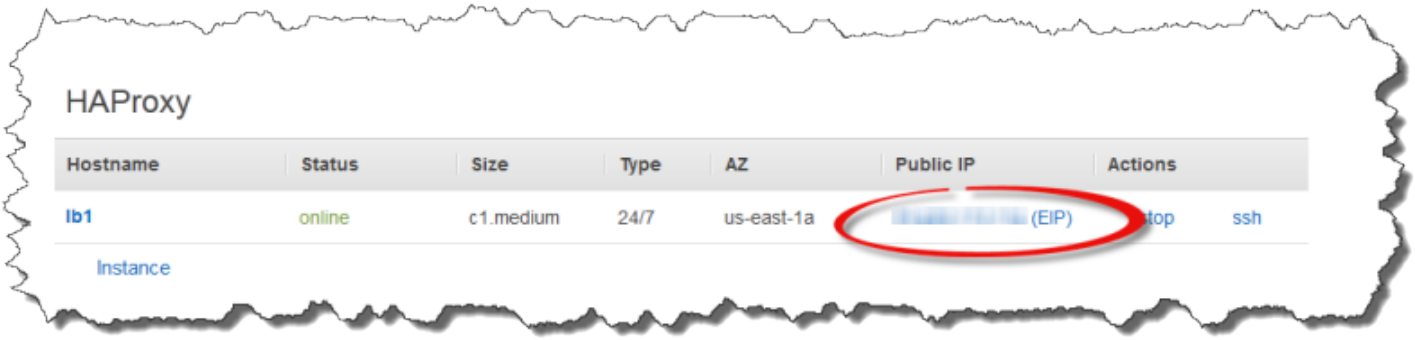
- Periodically runs the specified health check method on each application server instance to assess its health.

  If the method does not return within a specified timeout period, the instance is presumed to have failed and HAProxy stops routing requests to the instance. OpsWorks Stacks also provides a way to automatically replace failed instances. For more information, see Using Auto Healing. You can change the health check method when you create the layer.

- Collects statistics and optionally displays them on a web page.

> **Important**
>
> For health check to work correctly with the default OPTIONS method, your app must return a 2xx or 3xx status code.

By default, when you add an instance to a HAProxy layer, OpsWorks Stacks assigns it an Elastic IP address to represent the application, which is public to the world. Because the HAProxy instance's Elastic IP address is the application's only publicly exposed URL, you don't have to create and manage public domain names for the underlying application server instances. You can obtain the address by going to the **Instances** page and examining the instance's public IP address, as the following illustration shows. An address that is followed by (EIP) is an Elastic IP address. For more information on Elastic IP addresses, see Elastic IP Addresses (EIP).



When you stop an HAProxy instance, OpsWorks Stacks retains the Elastic IP address and reassigns it to the instance when you restart it. If you delete an HAProxy instance, by default, OpsWorks Stacks deletes the instance's IP address. To retain the address, clear the **Delete instance's Elastic IP** option, as shown in the following illustration.



This option affects what happens when you add a new instance to the layer to replace a deleted instance:

- If you retained the deleted instance's Elastic IP address, OpsWorks Stacks assigns the address to the new instance.
- Otherwise, OpsWorks Stacks assigns a new Elastic IP address to the instance and you must update your DNS registrar settings to map to the new address.

When application server instances come on line or go off line—either manually or as a consequence of automatic scaling or auto healing—the load balancer configuration must be updated to route traffic to the current set of online instances. This task is handled automatically by the layer's built-in recipes:

- When new instances come on line, OpsWorks Stacks triggers a Configure lifecycle event. The HAProxy layer's built-in Configure recipes update the load balancer configuration so that it also distributes requests to any new application server instances.
- When instances go off line or an instance fails a health check, OpsWorks Stacks also triggers a Configure lifecycle event. The HAProxy Configure recipes update the load balancer configuration to route traffic to only the remaining online instances.

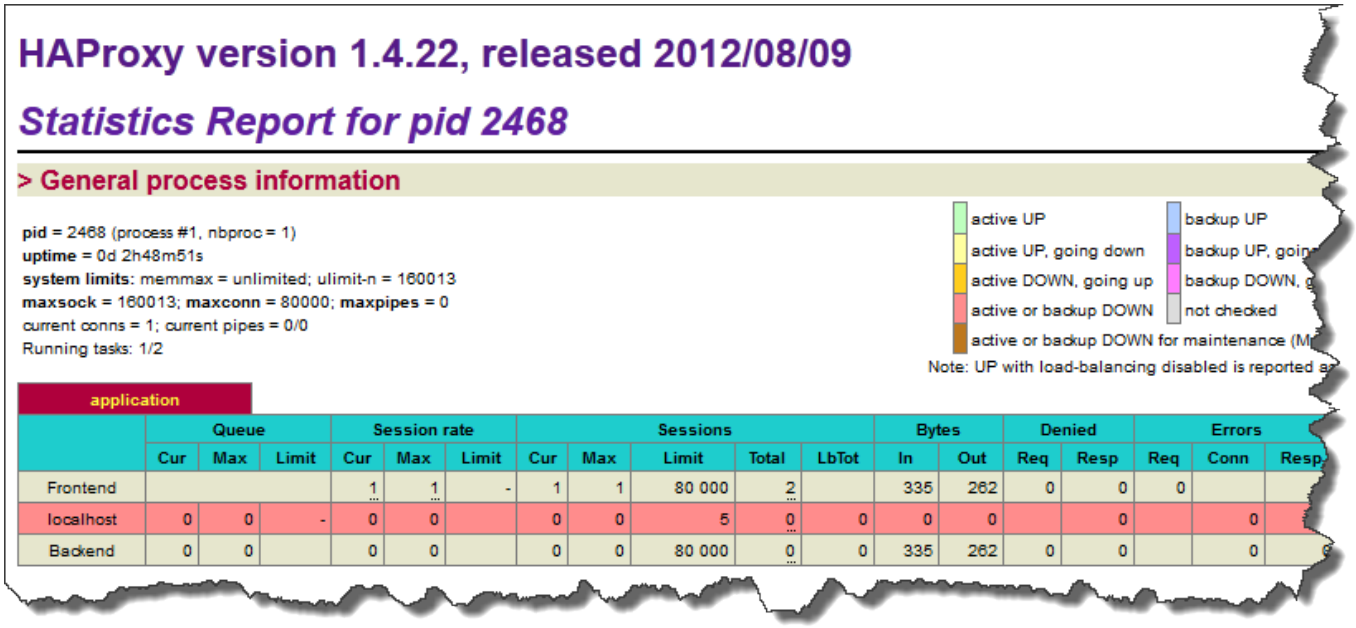
Finally, you can also use a custom domain with the HAProxy layer. For more information, see Using Custom Domains.

Statistics Page

If you have enabled the statistics page, the HAProxy displays a page containing a variety of metrics at the specified URL.

To view HAProxy statistics

1. Obtain the HAProxy instance's **Public DNS** name from the instance's **Details** page and copy it.
2. On the **Layers** page, click **HAProxy** to open the layer's details page.
3. Obtain the statistics URL from the layer details and append it to the Public DNS name. For example: **`http://ec2-54-245-102-172.us-west-2.compute.amazonaws.com/haproxy?stats`**. to it.
4. Paste the URL from the previous step into your browser and use the user name and password that you specified when you created the layer to open the statistics page.



MySQL Layer Reference

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

This layer is available only for Linux-based stacks.

The MySQL layer supports MySQL, a widely used relational database management system. OpsWorks Stacks installs the most recent available version, which depends on the operating system. If you add a MySQL instance, the needed access information is provided to the application server layers. You must write custom Chef recipes to set up master–master or master–slave configurations.

Short name: db-master

Compatibility: A MySQL layer is compatible with the following layers: custom, lb, memcached, monitoring-master, nodejs-app, php-app, rails-app, and web.

Open ports: A MySQL layer allows public access to port 22(SSH) and all ports from the stack's web servers, custom servers, and Rails, PHP, and Node.js application servers.

Autoassign Elastic IP addresses: Off by default

Default EBS volume: Yes, at `/vol/mysql`

Default security group: AWS-OpsWorks-DB-Master-Server

Configuration: To configure a MySQL layer, you must specify the following:

- Root user password
- MySQL engine

Setup recipes:

- opsworks_initial_setup
- ssh_host_keys
- ssh_users
- mysql::client
- dependencies
- ebs
- opsworks_ganglia::client
- mysql::server
- dependencies
- deploy::mysql

Configure recipes:

- opsworks_ganglia::configure-client
- ssh_users

- agent_version
- deploy::mysql

Deploy recipes:

- deploy::default
- deploy::mysql

Shutdown recipes:

- opsworks_shutdown::default
- mysql::stop

Installation:

- OpsWorks Stacks uses the instance's package installer to install MySQL and its log files to their default locations. For more information, see MySQL Documentation.

MySQL OpsWorks Layer

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This layer is available only for Chef 11 and earlier Linux-based stacks.

A MySQL OpsWorks layer provides a blueprint for Amazon EC2 instances that function as a MySQL database master. A built-in recipe creates a database for each application that has been deployed

to an application server layer. For example, if you deploy a PHP application “myapp,” the recipe creates a “myapp” database.

The MySQL layer has the following configuration settings.

MySQL root user password

(Required) The root user password.

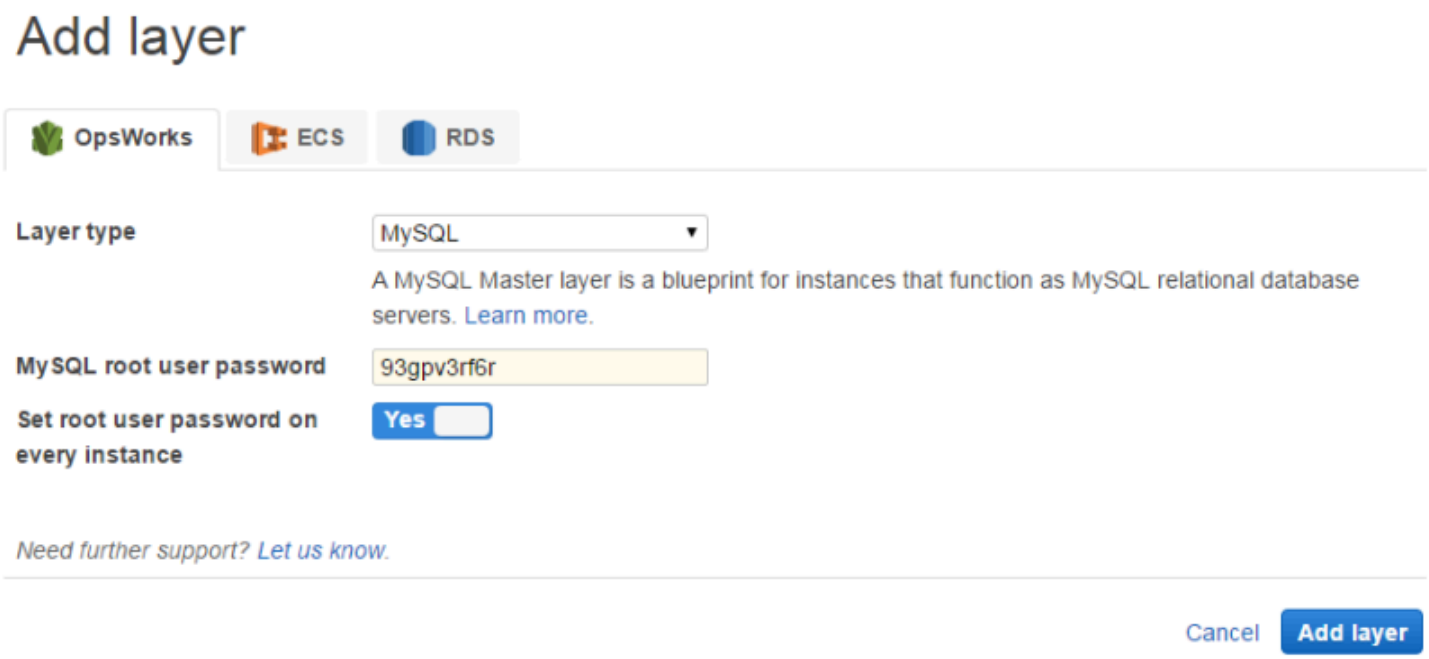
Set root user password on every instance

(Optional) Whether the root user password is included in the stack configuration and deployment attributes that are installed on every instance in the stack. The default setting is **Yes**.

If you set this value to **No**, OpsWorks Stacks passes the root password only to application server instances.

Custom security groups

(Optional) A custom security group to be associated with the layer. For more information, see Create a New Stack.



You can add one or more instances to the layer, each of which represents a separate MySQL database master. You can then attach an instance to an app, which installs the necessary

connection information on the app's application servers. The application can then use the connection information to connect to the instance's database server.

Application Server Layers Reference

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks supports several different application and static web page servers.

Topics

- AWS Flow (Ruby) Layer Reference
- Java App Server Layer Reference
- Node.js App Server Layer Reference
- PHP App Server Layer Reference
- Rails App Server Layer Reference
- Static Web Server Layer Reference

AWS Flow (Ruby) Layer Reference

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This layer is available only for Linux-based stacks.

An AWS Flow (Ruby) layer provides a blueprint for instances that host Amazon Simple Workflow Service activity and workflow workers.

Short name: aws-flow-ruby

Compatibility: An AWS Flow (Ruby) layer is compatible with PHP App Server, MySQL, Memcached, Ganglia, and custom layers.

Open ports: None.

IAM role: aws-opsworks-ec2-role-with-swf is the standard AWS Flow (Ruby) role that OpsWorks Stacks creates for you, if requested.

Autoassign Elastic IP addresses: Off by default

Default EBS Volume: No

Default security group: AWS-OpsWorks-AWS-Flow-Ruby-Server

Setup recipes:

- opsworks_initial_setup
- ssh_host_keys
- ssh_users
- mysql::client
- dependencies
- ebs
- opsworks_ganglia::client
- opsworks_aws_flow_ruby::setup

Configure recipes:

- opsworks_ganglia::configure-client
- ssh_users

- mysql::client
- agent_version
- opsworks_aws_flow_ruby::configure

Deploy recipes:

- deploy::default
- deploy::aws-flow-ruby

Undeploy recipes:

- deploy::aws-flow-ruby-undeploy

Shutdown recipes:

- opsworks_shutdown::default

Java App Server Layer Reference

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This layer is available only for Linux-based stacks.

A Java App Server layer supports an Apache Tomcat 7.0 application server.

Short name: java-app

Compatibility: A Java App Server layer is compatible with the following layers: custom, db-master, and memcached.

Open ports: A Java App Server layer allows public access to ports 22 (SSH), 80 (HTTP), 443 (HTTPS), and all ports from load balancers.

Autoassign Elastic IP addresses: Off by default

Default EBS Volume: No

Default security group: AWS-OpsWorks-Java-App-Server

Setup recipes:

- opsworks_initial_setup
- ssh_host_keys
- ssh_users
- mysql::client
- dependencies
- ebs
- opsworks_ganglia::client
- opsworks_java::setup

Configure recipes:

- opsworks_ganglia::configure-client
- ssh_users
- agent_version
- opsworks_java::configure

Deploy recipes:

- deploy::default
- deploy::java

Undeploy recipes:

- deploy::java-undeploy

Shutdown recipes:

- opsworks_shutdown::default
- deploy::java-stop

Installation:

- Tomcat installs to `/usr/share/tomcat7`.
- For more information about how to produce log files, see Logging in Tomcat.

Node.js App Server Layer Reference

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This layer is available only for Linux-based stacks.

A Node.js App Server layer supports a Node.js application server, which is a platform for implementing highly scalable network application servers. Programs are written in JavaScript, using event-driven asynchronous I/O to minimize overhead and maximize scalability.

Short name: nodejs-app

Compatibility: A Node.js App Server layer is compatible with the following layers: custom, db-master, memcached, and monitoring-master.

Open ports: A Node.js App Server layer allows public access to ports 22 (SSH), 80 (HTTP), 443 (HTTPS), and all ports from load balancers.

Autoassign Elastic IP addresses: Off by default

Default EBS volume: No

Default security group: AWS-OpsWorks-nodejs-App-Server

Setup recipes:

- opsworks_initial_setup
- ssh_host_keys
- ssh_users
- mysql::client
- dependencies
- ebs
- opsworks_ganglia::client
- opsworks_nodejs
- opsworks_nodejs::npm

Configure recipes:

- opsworks_ganglia::configure-client
- ssh_users
- agent_version
- opsworks_nodejs::configure

Deploy recipes:

- deploy::default
- opsworks_nodejs
- opsworks_nodejs::npm
- deploy::nodejs

Undeploy recipes:

- deploy::nodejs-undeploy

Shutdown recipes:

- opsworks_shutdown::default
- deploy::nodejs-stop

Installation:

- Node.js installs to `/usr/local/bin/node`.
- For more information about how to produce log files, see How to log in node.js on the Nodejitsu website.

Node.js application configuration:

- The main file run by Node.js must be named `server.js` and reside in the root directory of the deployed application.
- The Node.js application must be set to listen on port 80 (or port 443, if applicable).

> **Note**
>
> Node.js apps that run Express commonly use the following code to set the listening port, where `process.env.PORT` represents the default port and resolves to 80:
>
> ```
> app.set('port', process.env.PORT || 3000);
> ```
>
> With OpsWorks Stacks, you must explicitly specify port 80, as follows:
>
> ```
> app.set('port', 80);
> ```

PHP App Server Layer Reference

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This layer is available only for Linux-based stacks.

The PHP App Server layer supports a PHP application server by using Apache2 with mod_php.

Short name: php-app

Compatibility: A PHP App Server layer is compatible with the following layers: custom, db-master, memcached, monitoring-master, and rails-app.

Open ports: A PHP App Server layer allows public access to ports 22 (SSH), 80 (HTTP), 443 (HTTPS), and all ports from load balancers.

Autoassign Elastic IP addresses: Off by default

Default EBS volume: No

Default security group: AWS-OpsWorks-PHP-App-Server

Setup recipes:

- opsworks_initial_setup
- ssh_host_keys
- ssh_users
- mysql::client

- dependencies
- ebs
- opsworks_ganglia::client
- mysql::client
- dependencies
- mod_php5_apache2

Configure recipes:

- opsworks_ganglia::configure-client
- ssh_users
- agent_version
- mod_php5_apache2::php
- php::configure

Deploy recipes:

- deploy::default
- deploy::php

Undeploy recipes:

- deploy::php-undeploy

Shutdown recipes:

- opsworks_shutdown::default
- apache2::stop

Installation:

- OpsWorks Stacks uses the instance's package installer to install Apache2, mod_php and the associated log files to their default locations. For more information about installation, see Apache. For more information about logging, see Log Files.

Rails App Server Layer Reference

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This layer is available only for Linux-based stacks.

The Rails App Server layer supports a Ruby on Rails application server.

Short name: rails-app

Compatibility: A Rails App Server layer is compatible with the following layers: custom, db-master, memcached, monitoring-master, php-app.

Ports: A Rails App Server layer allows public access to ports 22(SSH), 80 (HTTP), 443 (HTTPS), and all ports from load balancers.

Autoassign Elastic IP addresses: Off by default

Default EBS volume: No

Default security group: AWS-OpsWorks-Rails-App-Server

Configuration: To configure a Rails App Server layer, you must specify the following:

- Ruby version
- Rails stack
- Rubygems version
- Whether to install and manage Bundler

- The Bundler version

Setup recipes:

- opsworks_initial_setup
- ssh_host_keys
- ssh_users
- mysql::client
- dependencies
- ebs
- opsworks_ganglia::client
- apache2 apache2::mod_deflate
- passenger_apache2
- passenger_apache2::mod_rails
- passenger_apache2::rails

Configure recipes:

- opsworks_ganglia::configure-client
- ssh_users
- agent_version
- rails::configure

Deploy recipes:

- deploy::default
- deploy::rails

Undeploy recipes:

- deploy::rails-undeploy

Shutdown recipes:

- opsworks_shutdown::default
- apache2::stop

Installation:

- OpsWorks Stacks uses the instance's package installer to install Apache2 with mod_passenger, mod_rails, and the associated log files to their default locations. For more information about installation, see Phusion Passenger. For more information about logging, see Log Files.

Static Web Server Layer Reference

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This layer is available only for Linux-based stacks.

The Static Web Server layer serves static HTML pages, which can include client-side code such as JavaScript. It is based on Nginx, which is an open source HTTP, reverse proxy, and mail proxy server.

Short name: web

Compatibility: A Static Web Server layer is compatible with the following layers: custom, db-master, memcached.

Open ports: A Static Web Server layer allows public access to ports 22(SSH), 80 (HTTP), 443 (HTTPS), and all ports from load balancers.

Autoassign Elastic IP addresses: Off by default

Default EBS volume: No

Default security group: AWS-OpsWorks-Web-Server

Setup recipes:

- opsworks_initial_setup
- ssh_host_keys
- ssh_users
- mysql::client
- dependencies
- ebs
- opsworks_ganglia::client
- nginx

Configure recipes:

- opsworks_ganglia::configure-client
- ssh_users
- agent_version

Deploy recipes:

- deploy::default
- deploy::web

Undeploy recipes:

- deploy::web-undeploy

Shutdown recipes:

- opsworks_shutdown::default
- nginx::stop

Installation:

- Nginx installs to `/usr/sbin/nginx`.
- Nginx log files are in `/var/log/nginx`.

Application Server Layers

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> These layers are available only for Chef 11 and earlier Linux-based stacks.

OpsWorks Stacks supports several different application servers, where "application" includes static web pages. Each type of server has a separate OpsWorks Stacks layer, with built-in recipes that handle installing the application server and any related packages on each of the layer's instances, deploying apps, and so on. For example, the Java App Server layer installs several packages—including Apache, Tomcat, and OpenJDK—and deploys Java apps to each of the layer's instances.

The following is the basic procedure for using an application server layers:

1. Create one of the available **App Server** layer types.
2. Add one or more instances to the layer.
3. Create apps and deploy them to the instances. For more information, see Apps.
4. (Optional) If the layer has multiple instances, you can add a load balancer, which distributes incoming traffic across the instances. For more information, see HAProxy OpsWorks Stacks Layer.

Topics

- AWS Flow (Ruby) Layer

- Java App Server OpsWorks Stacks Layer
- Node.js App Server OpsWorks Stacks Layer
- PHP App Server OpsWorks Stacks Layer
- Rails App Server OpsWorks Stacks Layer
- Static Web Server OpsWorks Stacks Layer

AWS Flow (Ruby) Layer

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This layer is available only for Linux-based stacks.

An AWS Flow (Ruby) layer is an OpsWorks Stacks layer that provides a blueprint for instances that host Amazon SWF activity and workflow workers. The workers are implemented by using the AWS Flow Framework for Ruby, which is a programming framework that simplifies the process of implementing a distributed asynchronous application while providing all the benefits of Amazon SWF. It is ideal for implementing applications to address a broad range of scenarios, including business processes, media encoding, long-running tasks, and background processing.

The AWS Flow (Ruby) layer includes the following configuration settings.

RubyGems version

The framework's Gem version.

Bundler version

The Bundler version.

EC2 Instance profile

A user-defined Amazon EC2 instance profile to be used by the layer's instances. This profile must grant permissions for applications running on the layer's instances to access Amazon SWF.

If your account does not have an appropriate profile, you can select **New profile with SWF access** to have OpsWorks Stacks update the profile for or you can update it yourself by using the IAM console. You can then use the updated profile for all subsequent AWS Flow layers. The following is a brief description of how to create the profile by using the IAM console. For more information, see Identity and Access Management in Amazon Simple Workflow Service.

Creating a profile for AWS Flow (Ruby) instances

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Policies** in the navigation pane and choose **Create policy** to create a new customer-managed policy.
3. For **Service**, choose **SWF**.
4. For **Actions**, choose **All SWF actions (swf:*)**.
5. For **Amazon Resource Name (ARN)**, enter the ARN that specifies which Amazon SWF domains the workers can access. Choose **`All resources`** to provide access to all domains.
6. Choose **Next**.
7. Optionally, enter a tag to identify the policy.
8. Choose **Next**.
9. When you are finished, choose **Create policy**.
10. Choose **Roles** in the navigation pane and choose **Create role**.
11. Specify the role name and choose **Next Step**. You cannot change the name after the role has been created.
12. Choose **AWS service** and then choose **EC2**.
13. Choose **Next**.
14. From the **Permissions policies** list, choose the policy that you created earlier.
15. Choose **Next**.
16. Enter a role name and choose **Create role**. You cannot change the name after the role has been created.
17. Specify this profile when you create an AWS Flow (Ruby) layer in OpsWorks Stacks.

Java App Server OpsWorks Stacks Layer

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This layer is available only for Linux-based stacks.

The Java App Server layer is an OpsWorks Stacks layer that provides a blueprint for instances that function as Java application servers. This layer is based on Apache Tomcat 7.0 and Open JDK 7. OpsWorks Stacks also installs the Java connector library, which allows Java apps to use a JDBC `DataSource` object to connect to a back end data store.

Installation: Tomcat is installed in `/usr/share/tomcat7`.

The **Add Layer** page provides the following configuration options:

Java VM Options

You can use this setting to specify custom Java VM options; there are no default options. For example, a common set of options is `-Djava.awt.headless=true -Xmx128m -XX:+UseConcMarkSweepGC`. If you use **Java VM Options**, make sure that you pass a valid set of options; OpsWorks Stacks does not validate the string. If you attempt to pass an invalid option, the Tomcat server typically fails to start, which causes setup to fail. If that happens, you can examine the instance's setup Chef log for details. For more information on how to view and interpret Chef logs, see Chef Logs.

Custom security groups

This setting appears if you chose to not automatically associate a built-in OpsWorks Stacks security group with your layers. You must specify which security group to associate with the layer. For more information, see Create a New Stack.

Elastic Load Balancer

You can attach an Elastic Load Balancing load balancer to the layer's instances. For more information, see Elastic Load Balancing Layer.

You can specify other configuration settings by using custom JSON or a custom attributes file. For more information, see Custom Configuration.

> **Important**
>
> If your Java application uses SSL, we recommend that you disable SSLv3 if possible to address the vulnerabilities described in CVE-2014-3566. For more information, see Disabling SSLv3 for Apache Servers.

Topics

- Disabling SSLv3 for Apache Servers
- Custom Configuration
- Deploying Java Apps

Disabling SSLv3 for Apache Servers

To disable SSLv3, you must modify the Apache server's `ssl.conf` file's `SSLProtocol` setting. To do so, you must override the built-in apache2 cookbook's `ssl.conf.erb` template file, which the Java App Server layer's Setup recipes use to create `ssl.conf`. The details depend on which operating system you specify for the layer's instances. The following summarizes the required modifications for Amazon Linux and Ubuntu systems. SSLv3 is automatically disabled for Red Hat Enterprise Linux (RHEL) systems. For more information on how to override a built-in template, see Using Custom Templates.

Amazon Linux

The `ssl.conf.erb` file for these operating systems is in the `apache2` cookbook's `apache2/templates/default/mods` directory. The following shows the relevant part of the built-in file.

```
...
#SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL

# enable only secure protocols: SSLv3 and TLSv1.2, but not SSLv2
SSLProtocol all -SSLv2
</IfModule>
```

Override `ssl.conf.erb` and modify the `SSLProtocol` setting as follows.

```
...
#SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL

# enable only secure protocols: SSLv3 and TLSv1.2, but not SSLv2
SSLProtocol all -SSLv3 -SSLv2
</IfModule>
```

Ubuntu 14.04 LTS

The `ssl.conf.erb` file for this operating system is in the `apache2` cookbook's `apache2/templates/ubuntu-14.04/mods` directory. The following shows the relevant part of the built-in file.

```
...
# The protocols to enable.
# Available values: all, SSLv3, TLSv1.2
# SSL v2 is no longer supported
SSLProtocol all
...
```

Change this setting to the following.

```
...
# The protocols to enable.
# Available values: all, SSLv3, TLSv1.2
# SSL v2 is no longer supported
SSLProtocol all -SSLv3 -SSLv2
...
```

Custom Configuration

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks exposes additional configuration settings as built-in attributes, which are all in the `opsworks_java` namespace. You can use custom JSON or a custom attributes file to override the built-in attributes and specify custom values. For example, the JVM and Tomcat versions are represented by the built-in `jvm_version` and `java_app_server_version` attributes, both of which are set to 7. You can use custom JSON or a custom attributes file to set either or both to 6. The following example uses custom JSON to set both attributes to 6:

```
{
  "opsworks_java": {
    "jvm_version": 6,
    "java_app_server_version" : 6
  }
}
```

For more information, see Using Custom JSON.

Another example of custom configuration is installing a custom JDK by overriding the `use_custom_pkg_location`, `custom_pkg_location_url_debian`, and `custom_pkg_location_url_rhel` attributes.

> **Note**
>
> If you override the built-in cookbooks, you will need to update those components yourself.

For more information on attributes and how to override them, see Overriding Attributes. For a list of built in attributes, see opsworks_java Attributes.

Deploying Java Apps

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The following topics describe how to deploy apps to a Java App Server layer's instances. The examples are for JSP apps, but you can use essentially the same procedure for installing other types of Java app.

You can deploy JSP pages from any of the supported repositories. If you want to deploy WAR files, note that OpsWorks Stacks automatically extracts WAR files that are deployed from an Amazon S3 or HTTP archive, but not from a Git or Subversion repository. If you want to use Git or Subversion for WAR files, you can do one of the following:

- Store the extracted archive in the repository.
- Store the WAR file in the repository and use a Chef deployment hook to extract the archive, as described in the following example.

You can use Chef deployment hooks to run user-supplied Ruby applications on an instance at any of four deployment stages. The application name determines the stage. The following is an example of a Ruby application named `before_migrate.rb`, which extracts a WAR file that has been deployed from a Git or Subversion repository. The name associates the application with the Checkout deployment hook so it runs at the beginning of the deployment operation, after the code is checked but before migration. For more information on how to use this example, see Using Chef Deployment Hooks.

```
::Dir.glob(::File.join(release_path, '*.war')) do |archive_file|
  execute "unzip_#{archive_file}" do
    command "unzip #{archive_file}"
    cwd release_path
  end
```

```
end
```

Note

When you deploy an update to a JSP app, Tomcat might not recognize the update and instead continue to run the existing app version. This can happen, for example, if you deploy your app as a .zip file that contains only a JSP page. To ensure that Tomcat runs the most recently deployed version, the project's root directory should include a WEB-INF directory that contains a `web.xml` file. A `web.xml` file can contain a variety of content, but the following is sufficient to ensure that Tomcat recognizes updates and runs the currently deployed app version. You don't have to change the version for each update. Tomcat will recognize the update even if the version hasn't changed.

```
<context-param>
  <param-name>appVersion</param-name>
  <param-value>0.1</param-value>
</context-param>
```

Topics

- Deploying a JSP App
- Deploying a JSP App with a Back-End Database

Deploying a JSP App

To deploy a JSP app, specify the name and repository information. You can also optionally specify domains and SSL settings. For more information on how to create an app, see Adding Apps. The following procedure shows how to create and deploy a simple JSP page from a public Amazon S3 archive. For information on how to use other repository types, including private Amazon S3 archives, see Application Source.

The following example shows the JSP page, which simply displays some system information.

```
<%@ page import="java.net.InetAddress" %>
<html>
```

```
<body>
<%
    java.util.Date date = new java.util.Date();
    InetAddress inetAddress = InetAddress.getLocalHost();
%>
The time is
<%
    out.println( date );
    out.println("<br>Your server's hostname is "+inetAddress.getHostName());
%>
<br>
</body>
</html>
```

Note

The following procedure assumes that you are already familiar with the basics of creating stacks, adding instances to layers, and so on. If you are new to OpsWorks Stacks, you should first see Getting Started with Chef 11 Linux Stacks.

To deploy a JSP page from an Amazon S3 archive

1. Create a stack with a Java App Server layer, add a 24/7 instance to the layer, and start it.
2. Copy the code to a file named `simplejsp.jsp`, put the file in a folder named `simplejsp`, and create a `.zip` archive of the folder. The names are arbitrary; you can use any file or folder names that you want. You can also use other types of archive, including gzip, bzip2, tarball, or Java WAR file. Note that OpsWorks Stacks does not support uncompressed tarballs. To deploy multiple JSP pages, include them in the same archive.
3. Upload the archive to an Amazon S3 bucket and make the file public. Copy the file's URL for later use . For more information on how to create buckets and upload files, go to Get Started With Amazon Simple Storage Service.
4. Add an app to the stack and specify following settings:
   - **Name** – `SimpleJSP`
   - **App type** – `Java`
   - **Repository type** – `Http Archive`
   - **Repository URL** – the Amazon S3 URL of your archive file.

Use the default values for the remaining settings and then click **Add App** to create the app.

5. Deploy the app to the Java App Server instance.

You can now go to the app's URL and view the app. If you have not specified a domain, you can construct a URL by using either the instance's public IP address or its public DNS name. To get an instance's public IP address or public DNS name, go the OpsWorks Stacks console and click the instance's name on the **Instances** page to open its details page.

The rest of URL depends on the app's short name, which is a lowercase name that OpsWorks Stacks generates from the app name that you specified when you created the app. For example the short name of SimpleJSP is simplejsp. You can get an app's short name from its details page.

- If the short name is `root`, you can use either `http://`*`public_DNS`*`/`*`appname`*`.jsp` or `http://`*`public_IP`*`/`*`appname`*`.jsp`.
- Otherwise, you can use either `http://`*`public_DNS`*`/`*`app_shortname`*`/`*`appname`*`.jsp` or `http://`*`public_IP`*`/`*`app_shortname`*`/`*`appname`*`.jsp`.

If you have specified a domain for the app, the URL is `http://`*`domain`*`/`*`appname`*`.jsp`.

The URL for the example would be something like `http://192.0.2.0/simplejsp/simplejsp.jsp`.

If you want to deploy multiple apps to the same instance, you should not use `root` as a short name. This can cause URL conflicts that prevent the app from working properly. Instead, assign a different domain name to each app.

Deploying a JSP App with a Back-End Database

JSP pages can use a JDBC `DataSource` object to connect to a back end database. You create and deploy such an app by using the procedure in the previous section, with one additional step to set up the connection.

The following JSP page shows how to connect to a `DataSource` object.

```
<html>
  <head>
    <title>DB Access</title>
```

```
  </head>
  <body>
    <%@ page language="java" import="java.sql.*,javax.naming.*,javax.sql.*" %>
    <%
      StringBuffer output = new StringBuffer();
      DataSource ds = null;
      Connection con = null;
      Statement stmt = null;
      ResultSet rs = null;
      try {
        Context initCtx = new InitialContext();
        ds = (DataSource) initCtx.lookup("java:comp/env/jdbc/mydb");
        con = ds.getConnection();
        output.append("Databases found:<br>");
        stmt = con.createStatement();
        rs = stmt.executeQuery("show databases");
        while (rs.next()) {
          output.append(rs.getString(1));
          output.append("<br>");
        }
      }
      catch (Exception e) {
        output.append("Exception: ");
        output.append(e.getMessage());
        output.append("<br>");
      }
      finally {
        try {
          if (rs != null) {
            rs.close();
          }
          if (stmt != null) {
            stmt.close();
          }
          if (con != null) {
            con.close();
          }
        }
        catch (Exception e) {
          output.append("Exception (during close of connection): ");
          output.append(e.getMessage());
          output.append("<br>");
        }
      }
```

```
    %>
    <%= output.toString() %>
  </body>
</html>
```

OpsWorks Stacks creates and initializes the `DataSource` object, binds it to a logical name, and registers the name with a Java Naming and Directory Interface (JNDI) naming service. The complete logical name is `java:comp/env/`*`user-assigned-name`*. You must specify the user-assigned part of the name by adding custom JSON attributes to the stack configuration and deployment attributes to define the `['opsworks_java']['datasources']` attribute, as described in the following.

To deploy a JSP page that connects to a MySQL database

1. Create a stack with a Java App Server layer, add 24/7 instances to each layer, and start it.
2. Add a database layer to the stack. The details depend on which database you use.

   To use a MySQL instance for the example, add a MySQL layer to the stack, add a 24/7 instance to the layer, and start it.

   To use an Amazon RDS (MySQL) instance for the example:

   - Specify a MySQL database engine for the instance.
   - Assign the **AWS-OpsWorks-DB-Master-Server (*`security_group_id`*)** and **AWS-OpsWorks-Java-App-Server (*`security_group_id`*)** security groups to the instance. OpsWorks Stacks creates these security groups for you when you create your first stack in the region.
   - Create a database named `simplejspdb`.
   - Ensure that the master user name and password do not contain & or other characters that could cause a Tomcat error.

     In particular during startup Tomcat must parse the web app context file, which is an XML file that includes the master password and user name. If the either string includes a & character, the XML parser treats it as a malformed XML entity and throws a parsing exception, which prevents Tomcat from starting. For more information about the web app context file, see tomcat::context.
   - Add a MySQL driver to the Java App Server layer.

- Register the RDS instance with your stack.

  For more information about how to use Amazon RDS instances with OpsWorks Stacks, see Amazon RDS Service Layer.

3. Copy the example code to a file named `simplejspdb.jsp`, put the file in a folder named `simplejspdb`, and create a `.zip` archive of the folder. The names are arbitrary; you can use any file or folder names that you want. You can also use other types of archive, including gzip, bzip2, or tarball. To deploy multiple JSP pages, include them in the same archive. For information on how to deploy apps from other repository types, see Application Source.
4. Upload the archive to an Amazon S3 bucket and make the file public. Copy the file's URL for later use . For more information on how to create buckets and upload files, go to Get Started With Amazon Simple Storage Service.
5. Add an app to the stack and specify following settings:

   - **Name** – `SimpleJSPDB`
   - **App type** – `Java`
   - **Data source type** – **OpsWorks** (for a MySQL instance) or **RDS** (for an Amazon RDS instance).
   - **Database instance** – The MySQL instance you created earlier, which is typically named **db-master1(mysql)**, or the Amazon RDS instance, which will be named ***DB_instance_name*** **(mysql)**.
   - **Database name** – `simplejspdb`.
   - **Repository type** – `Http Archive`
   - **Repository URL** – the Amazon S3 URL of your archive file.

   Use the default values for the remaining settings and then click **Add App** to create the app.
6. Add the following custom JSON attributes to the stack configuration attributes, where simplejspdb is the app's short name.

```
{
  "opsworks_java": {
    "datasources": {
      "simplejspdb": "jdbc/mydb"
    }
  }
```

```
}
```

OpsWorks Stacks uses this mapping to generate a context file with the necessary database information.

For more information on how to add custom JSON attributes to the stack configuration attributes, see Using Custom JSON.

7. Deploy the app to the Java App Server instance.

You can now use the app's URL to view the app. For a description of how to construct the URL, see Deploying a JSP App.

The URL for the example would be something like `http://192.0.2.0/simplejspdb/simplejspdb.jsp`.

Note

The `datasources` attribute can contain multiple attributes. Each attribute is named with an apps short name and set to the appropriate user-assigned part of a logical name. If you have multiple apps, you can use separate logical names, which requires a custom JSON something like the following.

```
{
  "opsworks_java": {
    "datasources": {
      "myjavaapp": "jdbc/myappdb",
      "simplejsp": "jdbc/myjspdb",
      ...
    }
  }
}
```

Node.js App Server OpsWorks Stacks Layer

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This layer is available only for Linux-based stacks.

The Node.js App Server layer is an OpsWorks Stacks layer that provides a blueprint for instances that function as Node.js application servers. OpsWorks Stacks also installs Express, so the layer's instances support both standard and Express applications.

Installation: Node.js is installed in `/usr/local/bin/node`.

The **Add Layer** page provides the following configuration options:

Node.js version

For a list of currently supported versions, see OpsWorks Stacks operating systems.

Custom security groups

This setting appears if you chose to not automatically associate a built-in OpsWorks Stacks security group with your layers. You must specify which security group to associate with the layer. For more information, see Create a New Stack.

Elastic Load Balancer

You can attach an Elastic Load Balancing load balancer to the layer's instances.

Important

If your Node.js application uses SSL, we recommend that you disable SSLv3 if possible to address the vulnerabilities described in CVE-2015-8027. To do so, you must set **Node.js version** to `0.12.9`.

Deploying Node.js Apps

For a detailed walkthrough of how to implement a simple Node.js application for OpsWorks Stacks and deploy it to a stack, see Creating Your First Node.js Stack. In general, Node.js applications for OpsWorks Stacks should meet the following conditions:

- The main file must be named `server.js` and reside in the deployed application's root directory.
- Express apps must include a `package.json` file in the application's root directory.
- By default, the application must listen on port 80 (HTTP) or port 443 (HTTPS).

  It is possible to listen on other ports, but the Node.js App Server layer's built-in security group, **AWS-OpsWorks-nodejs-App-Server**, allows inbound user traffic only to ports 80, 443, and 22 (SSH). To allow inbound user traffic to other ports, create a security group with appropriate inbound rules and assign it to the Node.js App Server layer. Do not modify inbound rules by editing the built-in security group. Each time you create a stack, OpsWorks Stacks overwrites the built-in security groups with the standard settings, so any changes that you make will be lost.

Note

OpsWorks Stacks sets the PORT environment variable to 80 (default) or 443 (if you enable SSL), so you can use the following code to listen for requests.

```
app.listen(process.env.PORT);
```

If you configure a Node.js app to support SSL, you must specify the key and certificates. OpsWorks Stacks puts the data for each application server instance as separate files in the `/srv/www/`*`app_shortname`*`/shared/config` directory, as follows.

- `ssl.crt` – the SSL certificate.
- `ssl.key` – the SSL key.
- `ssl.ca` – the chain certificate, if you have specified one.

Your application can obtain the SSL key and certificates from those files.

PHP App Server OpsWorks Stacks Layer

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This layer is available only for Linux-based stacks.

The PHP App Server layer is an OpsWorks Stacks layer that provides a blueprint for instances that function as PHP application servers. The PHP App Server layer is based on Apache2 with `mod_php` and has no standard configuration options. The PHP and Apache version depends on which operating system you specify for the layer's instances.

Operating System	PHP Version	Apache Version
Amazon Linux 2018.03	5.3	2.2
Amazon Linux 2017.09	5.3	2.2
Amazon Linux 2017.03	5.3	2.2
Amazon Linux 2016.09	5.3	2.2
Amazon Linux 2016.03	5.3	2.2

Operating System	PHP Version	Apache Version
Amazon Linux 2015.09	5.3	2.2
Amazon Linux 2015.03	5.3	2.2
Amazon Linux 2014.09	5.3	2.2
Ubuntu 14.04 LTS	5.5	2.4

Installation: OpsWorks Stacks uses the instance's package installer to install Apache2 and `mod_php` in their default locations. For more information about installation, see Apache.

The **Add Layer** page provides the following configuration options:

Custom security groups

This setting appears if you chose to not automatically associate a built-in OpsWorks Stacks security group with your layers. You must specify which security group to associate with the layer. For more information, see Create a New Stack.

Elastic Load Balancer

You can attach an Elastic Load Balancing load balancer to the layer's instances.

You can modify some Apache configuration settings by using custom JSON or a custom attributes file. For more information, see Overriding Attributes. For a list of Apache attributes that can be overridden, see apache2 Attributes.

For an example of how to deploy a PHP App, including how to connect the app to a backend database, see Getting Started with Chef 11 Linux Stacks.

Important

If your PHP application uses SSL, we recommend that you disable SSLv3 if possible to address the vulnerabilities described in CVE-2014-3566. To do so, you must modify the `SSLProtocol` setting in the Apache server's `ssl.conf` file. For more information on how to modify this setting, see Disabling SSLv3 for Apache Servers.

Rails App Server OpsWorks Stacks Layer

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This layer is available only for Linux-based stacks.

The Rails App Server layer is an OpsWorks Stacks layer that provides a blueprint for instances that function as Rails application servers.

Installation: OpsWorks Stacks uses the instance's package installer to install the server packages in their default locations. For more information about Apache/Passenger installation, see Phusion Passenger. For more information about logging, see Log Files. For more information about Nginx/Unicorn installation, see Unicorn.

The **Add Layer** page provides the following configuration options, all of which are optional.

Ruby Version

The Ruby version that will be used by your applications. The default value is 2.3.

You can also specify your preferred Ruby version by overriding the `[:opsworks][:ruby_version]` attribute.

> **Note**
>
> OpsWorks Stacks installs a separate Ruby package to be used by recipes and the instance agent. For more information, see Ruby Versions.

Rails Stack

The default Rails stack is Apache2 with Phusion Passenger. You can also use Nginx with Unicorn.

> **Note**
>
> If you use Nginx and Unicorn, you must add the unicorn gem to your app's Gemfile, as in the following example:
>
> ```
> source 'https://rubygems.org'
> gem 'rails', '3.2.15'
> ...
> # Use unicorn as the app server
> gem 'unicorn'
> ...
> ```

Passenger Version

If you have specified Apache2/Passenger, you must specify the Passenger version. The default value is 5.0.28.

Rubygems Version

The default Rubygems version is 2.5.1

Install and Manage Bundler

Lets you choose whether to install and manage Bundler. The default value is **Yes**.

Bundler version

The default Bundler version is 1.12.5.

Custom security groups

This setting appears if you chose to not automatically associate a built-in OpsWorks Stacks security group with your layers. You must specify which security group to associate with the layer. For more information, see Create a New Stack.

Elastic Load Balancer

You can attach an Elastic Load Balancing load balancer to the layer's instances.

You can modify some configuration settings by using custom JSON or a custom attributes file. For more information, see Overriding Attributes. For a list of Apache, Nginx, Phusion Passenger, and Unicorn attributes that can be overridden, see Built-in Cookbook Attributes.

> **Important**
>
> If your Ruby on Rails application uses SSL, we recommend that you disable SSLv3 if possible to address the vulnerabilities described in CVE-2014-3566. For more information, see Disabling SSLv3 for Rails Servers.

Topics

- Disabling SSLv3 for Rails Servers
- Connecting to a Database
- Deploying Ruby on Rails Apps

Disabling SSLv3 for Rails Servers

To disable SSLv3 for Rails servers, update the layer's **Ruby Version** setting to 2.1 or higher, which installs Ruby 2.1.4 or higher as the version that applications use.

- Update the layer's **Ruby Version** setting to 2.1 or higher.
- Update the configuration file for your Rails stack, as follows.

Apache with Phusion Passenger

Update `SSLProtocol` setting in the Apache server's `ssl.conf` file, as described in Disabling SSLv3 for Apache Servers.

Nginx with Unicorn

Add an explicit `ssl_protocols` directive to the Nginx server's `nginx.conf` file. To disable SSLv3, override the built-in nginx cookbook's `nginx.conf.erb` template file, which the Rails App Server layer's Setup recipes use to create `nginx.conf`, and add the following directive:

```
ssl_protocols TLSv1.2;
```

For more information on how to configure `nginx.conf`, see Configuring HTTPS servers. For more information on how to override a built-in template, see Using Custom Templates.

Connecting to a Database

When you deploy an app, OpsWorks Stacks creates a new `database.yml` file using information from the app's `deploy` attributes. If you attach a MySQL or Amazon RDS instance to the app, OpsWorks Stacks adds the connection information to the `deploy` attributes, so that `database.yml` automatically contains the correct connection data.

If an app does not have an attached database, by default, OpsWorks Stacks does not add any connection information to the `deploy` attributes and does not create `database.yml`. If you want to use a different database, you can use custom JSON to add database attributes to the app's `deploy` attributes with the connection information. The attributes are all under`["deploy"]["`*`appshortname`*`"]["database"]`, where *`appshortname`* is the app's short name, which OpsWorks Stacks generates from the app name. The values you specify in custom JSON override any default settings. For more information, see Adding Apps.

OpsWorks Stacks incorporates the following `[:...][:database]` attribute values into `database.yml`. The required attributes depend on the particular database, but you must have a `host` attribute or OpsWorks Stacks will not create `database.yml`.

- `[:adapter] (String)` – The database adapter, such as `mysql`.
- `[:database]` (String) – The database name.
- `[:encoding]` (String) – The encoding, which is typically set to `utf8`.
- `[:host]` (String) – The host URL, such as `railsexample.cdlqlk5uwd0k.us-west-2.rds.amazonaws.com`.
- `[:reconnect]` (Boolean) – Whether the application should reconnect if the connection no longer exists.
- `[:password]` (String) – The database password.
- `[:port]` (Number). – The database's port number. Use this attribute to override the default port number, which is set by is set by the adapter.
- `[:username]` (String) – The database user name.

The following example shows custom JSON for an app whose short name is *myapp*.

```
{
  "deploy" : {
    "myapp" : {
      "database" : {
        "adapter" : "adapter",
        "database" : "databasename",
        "host" : "host",
        "password" : "password",
        "port" : portnumber
        "reconnect" : true/false,
        "username" : "username"
      }
    }
  }
}
```

For information on how to specify custom JSON, see Using Custom JSON. To see the template used to create `database.yml` (`database.yml.erb`), go to the built-in cookbook repository.

Deploying Ruby on Rails Apps

You can deploy Ruby on Rails apps from any of the supported repositories. The following shows how to deploy an example Ruby on Rails app to a server running an Apache/Passenger Rails stack. The example code is stored in a public GitHub repository, but the basic procedure is the same for the other supported repositories. For more information on how to create and deploy apps, see Apps. To view the example's code, which includes extensive comments, go to https://github.com/awslabs/opsworks-demo-rails-photo-share-app.

To deploy a Ruby on Rails app from a GitHub repository

1. Create a stack with a Rails App Server layer with Apache/Passenger as the Rails stack, add a 24/7 instance to the layer, and start it.
2. After the instance is online, add an app to the stack and specify following settings:

   - **Name** – Any name you prefer; the example uses `PhotoPoll`.

     OpsWorks Stacks uses this name for display purposes, and generates a short name for internal use and to identify the app in the stack configuration and deployment attributes. For example, the PhotoPoll short name is photopoll.
   - **App type** – **Ruby on Rails**.

- **Rails environment** – The available environments are determined by the application.

  The example app has three: **`development`**, **`test`**, and **`production`**. For this example, set the environment to **`development`**. See the example code for descriptions of each environment.

- **Repository type** – Any of the supported repository types. Specify `Git` for this example
- **Repository URL** – The repository that the code should be deployed from.

  For this example, set the URL to **`git://github.com/awslabs/opsworks-demo-rails-photo-share-app`**.

Use the default values for the remaining settings and then click **Add App** to create the app.

3. Deploy the app to the Rails App Server instance.
4. When deployment is finished, go to the **Instances** page and click the Rails App Server instance's public IP address. You should see the following:



Static Web Server OpsWorks Stacks Layer

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This layer is available only for Linux-based stacks.

The Static Web Server layer is an OpsWorks Stacks layer that provides a template for instances to serve static HTML pages, which can include client-side scripting. This layer is based on Nginx.

Installation: Nginx is installed in `/usr/sbin/nginx`.

The **Add Layer** page provides the following configuration options:

Custom security groups

This setting appears if you chose to not automatically associate a built-in OpsWorks Stacks security group with your layers. You must specify which security group to associate with the layer. For more information, see Create a New Stack.

Elastic Load Balancer

You can attach an Elastic Load Balancing load balancer to the layer's instances.

You can modify some Nginx configuration settings by using custom JSON or a custom attributes file. For more information, see Overriding Attributes. For a list of Apache attributes that can be overridden, see nginx Attributes.

Important

If your web application uses SSL, we recommend that you disable SSLv3 if possible to address the vulnerabilities described in CVE-2014-3566.
To disable SSLv3, you must modify the Nginx server's `nginx.conf` file. To do so, override the built-in nginx cookbook's `nginx.conf.erb` template file, which the Rails App Server layer's Setup recipes use to create `nginx.conf`, and add the following directive:

```
ssl_protocols TLSv1.2;
```

For more information on how to configure `nginx.conf`, see Configuring HTTPS servers.
For more information on how to override a built-in template, see Using Custom Templates.

ECS Cluster Layer Reference

Note

This layer is available only for Linux-based stacks.

An ECS Cluster layer represents an Amazon Elastic Container Service (Amazon ECS) cluster and simplifies cluster management.

Short name: ecs-cluster

Compatibility: An Amazon ECS service layer is compatible only with custom layers

Open ports: ECS Cluster allows public access to port 22 (SSH)

Autoassign Elastic IP addresses: Off by default

Default EBS volume: No

Default security group: AWS-OpsWorks-ECS-Cluster

Configuration: To configure an ECS Cluster layer, you must specify the following:

- Whether to assign public IP addresses or Elastic IP addresses to the container instances
- The instance profile for the container instances

Setup recipes:

- opsworks_initial_setup
- ssh_host_keys
- ssh_users
- mysql::client
- dependencies
- ebs
- opsworks_ganglia::client
- opsworks_ecs::setup

Configure recipes:

- opsworks_ganglia::configure-client
- ssh_users
- mysql::client
- agent_version
- opsworks_ecs::configure

Deploy recipes:

- deploy::default
- opsworks_ecs::deploy

Undeploy recipes:

- opsworks_ecs::undeploy

Shutdown recipes:

- opsworks_shutdown::default
- opsworks_ecs::shutdown

Installation:

- OpsWorks Stacks uses the instance's package installer to install Docker to its default locations
- The Chef log for the Setup event notes whether the Amazon ECS agent was successfully installed. Otherwise, the logs provided by OpsWorks Stacks do not include Amazon ECS error log information. For more information on how to handleAmazon ECS errors, see Amazon ECS Troubleshooting.

Custom Layer Reference

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

If the standard layers don't suit your requirements, you can create a custom layer. A stack can have multiple custom layers. By default, the custom layer runs a limited set of standard recipes that support basic functionality. You then implement the layer's primary functionality by implementing a set of custom Chef recipes for each of the appropriate lifecycle events to set up and configure the layer's software, and so on. Custom recipes run after the standard OpsWorks Stacks recipes for each event.

Short name: User-defined; each custom layer in a stack must have a different short name

Open ports: By default, a custom server layer opens public access to ports 22(SSH), 80 (HTTP), 443 (HTTPS), and all ports from the stack's Rails and PHP application server layers

Autoassign Elastic IP Addresses: Off by default

Default EBS volume: No

Default Security Group: AWS-OpsWorks-Custom-Server

Compatibility: Custom layers are compatible with the following layers: custom, db-master, lb, memcached, monitoring-master, nodejs-app, php-app, rails-app, and web

Configuration: To configure a custom layer, you must specify the following:

- The layer's name
- The layer's short name, which identifies the layer in Chef recipes and must use only a-z and numbers

For Linux stacks, the custom layer uses the following recipes.

Setup recipes:

- opsworks_initial_setup
- ssh_host_keys
- ssh_users
- mysql::client
- dependencies
- ebs
- opsworks_ganglia::client

Configure recipes:

- opsworks_ganglia::configure-client
- ssh_users
- agent_version

Deploy recipes:

- deploy::default

Shutdown recipes:

- opsworks_shutdown::default

Other Layers Reference

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks also supports the following layers.

Topics

- Ganglia Layer Reference
- Memcached Layer Reference

Ganglia Layer Reference

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This layer is available only for Linux-based stacks.

A Ganglia layer supports Ganglia, a distributed monitoring system that manages the storage and visualization of instance metrics. It is designed to work with hierarchical instance topologies, which makes it particularly useful for groups of instances. Ganglia has two basic components:

- A low-overhead client, which is installed on each instance in the stack and sends metrics to the master.
- A master, which collects metrics from the clients and stores them on an Amazon EBS volume. It also displays the metrics on a web page.

OpsWorks Stacks has a Ganglia monitoring agent on each instance that it manages. When you add a Ganglia layer to your stack and start it, the Ganglia agents on each instance report metrics to the Ganglia instance. To use Ganglia, add a Ganglia layer with one instance to the stack. You access the data by logging in to the Ganglia backend at the master's IP address. You can provide additional metric definitions by writing Chef recipes.

Short name: monitoring-master

Compatibility: A Ganglia layer is compatible with the following layers: custom, db-master, memcached, php-app, rails-app.

Open ports: Load-Balancer allows public access to ports 22(SSH), 80 (HTTP), and 443 (HTTPS).

Autoassign Elastic IP addresses: Off by default

Default EBS volume: Yes, at `/vol/ganglia`

Default security group: AWS-OpsWorks-Monitoring-Master-Server

Configuration: To configure a Ganglia layer, you must specify the following:

- The URI that provides access to the monitoring graphs. The default value is http://*DNSName*/ganglia, where *DNSName* is the Ganglia instance's DNS name.
- A user name and password that control access to the monitoring statistics.

Setup recipes:

- opsworks_initial_setup
- ssh_host_keys
- ssh_users
- mysql::client
- dependencies

- ebs
- opsworks_ganglia::client
- opsworks_ganglia::server

Configure recipes:

- opsworks_ganglia::configure-client
- ssh_users
- agent_version
- opsworks_ganglia::configure-server

Deploy recipes:

- deploy::default
- opsworks_ganglia::configure-server
- opsworks_ganglia::deploy

Shutdown recipes:

- opsworks_shutdown::default
- apache2::stop

Installation:

- The Ganglia client is installed under: `/etc/ganglia`.
- The Ganglia web front end is installed under: `/usr/share/ganglia-webfrontend`.
- The Ganglia logtailer is installed under: `/usr/share/ganglia-logtailer`.

Memcached Layer Reference

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate

their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

This layer is available only for Linux-based stacks.

Memcached is a distributed memory-caching system for arbitrary data. It speed up websites by caching strings and objects as keys and values in RAM to reduce the number of times an external data source must be read.

To use Memcached in a stack, create a Memcached layer and add one or more instances, which function as Memcached servers. The instances automatically install Memcached and the stack's other instances are able to access and use the Memcached servers. If you use a Rails App Server layer, OpsWorks Stacks automatically places a `memcached.yml` configuration file in the config directory of each instance in the layer. You can obtain the Memcached server and port number from this file.

Short name: memcached

Compatibility: A Memcached layer is compatible with the following layers: custom, db-master, lb, monitoring-master, nodejs-app, php-app, rails-app, and web.

Open ports: A Memcached layer allows public access to port 22(SSH) and all ports from the stack's web servers, custom servers, and Rails, PHP, and Node.js application servers.

Autoassign Elastic IP addresses: Off by default

Default EBS volume: No

Default security group: AWS-OpsWorks-Memcached-Server

To configure a Memcached layer, you must specify the cache size, in MB.

Setup recipes:

- opsworks_initial_setup

- ssh_host_keys
- ssh_users
- mysql::client
- dependencies
- ebs
- opsworks_ganglia::client
- memcached

Configure recipes:

- opsworks_ganglia::configure-client
- ssh_users
- agent_version

Deploy recipes:

- deploy::default

Shutdown recipes:

- opsworks_shutdown::default
- memcached::stop

Installation:

- OpsWorks Stacks uses the instance's package installer to install Memcached and its log files in their default locations.

Other Layers

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate

their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

These layers are available only for Chef 11 and earlier Linux-based stacks.

OpsWorks Stacks also supports the Ganglia and Memcached layers.

Topics

- Ganglia Layer
- Memcached

Ganglia Layer

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

This layer is available only for Chef 11 and earlier Linux-based stacks.

OpsWorks Stacks sends all of your instance and volume metrics to Amazon CloudWatch, making it easy to view graphs and set alarms to help you troubleshoot and take automated action based on the state of your resources. You can also use the Ganglia OpsWorks Stacks layer for additional application monitoring options such as storing your chosen metrics.

The Ganglia layer is a blueprint for an instance that monitors your stack by using Ganglia distributed monitoring. A stack usually has only one Ganglia instance. The Ganglia layer includes the following optional configuration settings:

Ganglia URL

The statistic's URL path. The complete URL is http://*DNSNameURLPath*, where *DNSName* is the associated instance's DNS name. The default *URLPath* value is "/ganglia" which corresponds to something like: http://ec2-54-245-151-7.us-west-2.compute.amazonaws.com/ganglia.

Ganglia user name

A user name for the statistics web page. You must provide the user name when you view the page. The default value is "opsworks".

Ganglia password

A password that controls access to the statistics web page. You must provide the password when you view the page. The default value is a randomly generated string.

> **Note**
>
> Record the password for later use; OpsWorks Stacks does not allow you to view the password after you create the layer. However, you can update the password by going to the layer's Edit page and clicking **Update password**.

Custom security groups

This setting appears if you chose to not automatically associate a built-in OpsWorks Stacks security group with your layers. You must specify which security group to associate with the layer. For more information, see Create a New Stack.

Elastic Load Balancer

You can attach an Elastic Load Balancing load balancer to the layer's instances.

> **Important**
>
> If your stack includes a Ganglia layer, we recommend that you disable SSLv3 if possible for that layer to address the vulnerabilities described in CVE-2014-3566. To do so, you

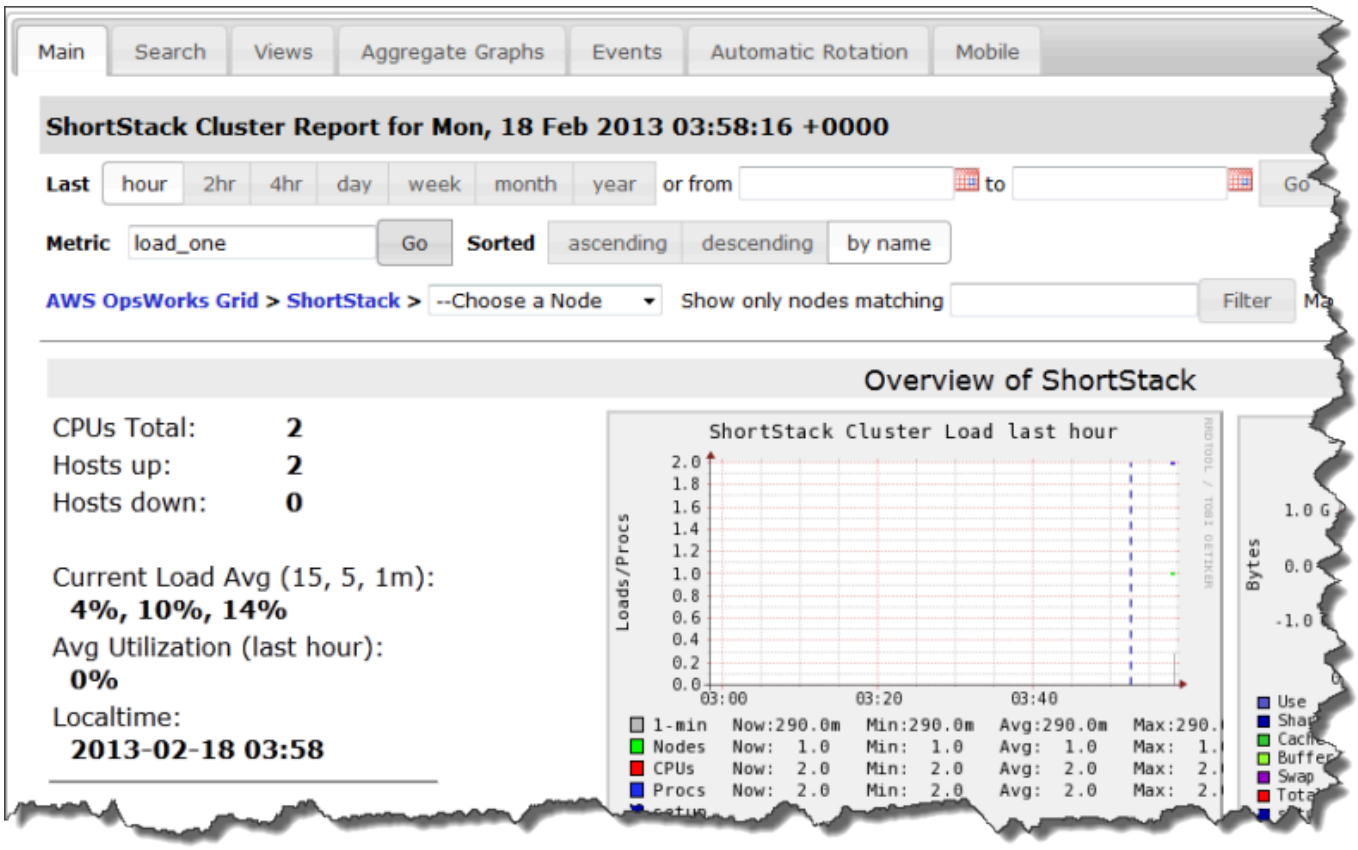
must override the Apache server's `ssl.conf.erb` template to modify the `SSLProtocol` setting. For details, see Disabling SSLv3 for Apache Servers.

View the Ganglia Statistics

OpsWorks Stacks recipes install a low-overhead Ganglia client on every instance. If your stack includes a Ganglia layer, the Ganglia client automatically starts reporting to the Ganglia as soon as the instance comes on line. The Ganglia uses the client data to compute a variety of statistics and displays the results graphically on its statistics web page.

To view Ganglia statistics

1. On the **Layers** page, click Ganglia to open the layer's details page.
2. In the navigation pane, click **Instances**. Under **Ganglia**, click the instance name.
3. Copy the instance's **Public DNS** name.
4. Use the DNS name to construct the statistics URL, which will look something like: http://ec2-54-245-151-7.us-west-2.compute.amazonaws.com/ganglia.
5. Paste the complete URL into your browser, navigate to the page, and enter the Ganglia user name and password to display the page. An example follows.



Memcached

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> This layer is available only for for Chef 11 and earlier Linux-based stacks.

A Memcached layer is an OpsWorks Stacks layer that provides a blueprint for instances that function as Memcached servers—a distributed memory-caching system for arbitrary data. The Memcached layer includes the following configuration settings.

Allocated memory (MB)

(Optional) The amount of cache memory (in MB) for each of the layer's instances. The default is 512 MB.

Custom security groups

This setting appears if you chose to not automatically associate a built-in OpsWorks Stacks security group with your layers. You must specify which security group to associate with the layer. For more information, see Create a New Stack.

Cookbook Components

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

A cookbook typically includes the following basic components:

- **Attribute** files contain a set of attributes that represent values to be used by the recipes and templates.
- **Template** files are templates that recipes use to create other files, such as configuration files.

  Template files typically let you modify the configuration file by overriding attributes—which can be done without touching the cookbook—instead of rewriting a configuration file. The standard practice is that whenever you expect to change a configuration file on an instance even slightly, you should use a template file.
- **Recipe** files are Ruby applications that define everything that is required to configure a system, including creating and configuring folders, installing and configuring packages, starting services, and so on.

Cookbooks don't have to have all three components. The simpler approaches to customization require only attribute or template files. In addition, cookbooks can optionally include other file types, such as definitions or specs.

This section describes the three standard cookbook components. For more information, especially about how to implement recipes, see Opscode.

Topics

- Attributes
- Templates
- Recipes

Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Recipes and templates depend on a variety of values, such as configuration settings. Rather than hardcode such values directly in recipes or templates, you can create an attribute file with an attribute that represents each value. You then use the attributes in your recipes or templates instead of explicit values. The advantage of using attributes is that you can override their values without touching the cookbook. For this reason, you should always use attributes to define the following types of values:

- Values that might vary from stack to stack or with time, such as user names.

  If you hardcode such values, you must change the recipe or template each time you need to change a value. By using attributes to define these values, you can use the same cookbooks for every stack and just override the appropriate attributes.
- Sensitive values, such as passwords or secret keys.

Putting explicit sensitive values in your cookbook can increase the risk of exposure. Instead, define attributes with dummy values and override them to set the actual values. The best way to override such attributes is with custom JSON. For more information, see Using Custom JSON.

For more information about attributes and how to override them, see Overriding Attributes.

The following example is a portion of an example attribute file.

```
...
default["apache"]["listen_ports"] = [ '80','443' ]
default["apache"]["contact"] = 'ops@example.com'
default["apache"]["timeout"] = 120
default["apache"]["keepalive"] = 'Off'
default["apache"]["keepaliverequests"] = 100
default["apache"]["keepalivetimeout"] = 3
default["apache"]["prefork"]["startservers"] = 16
default["apache"]["prefork"]["minspareservers"] = 16
default["apache"]["prefork"]["maxspareservers"] = 32
default["apache"]["prefork"]["serverlimit"] = 400
default["apache"]["prefork"]["maxclients"] = 400
default["apache"]["prefork"]["maxrequestsperchild"] = 10000
...
```

OpsWorks Stacks defines attributes by using the following syntax:

```
node.type["attribute"]["subattribute"]["..."]=value
```

You can also use colons (:), as follows:

```
node.type[:attribute][:subattribute][:...]=value
```

An attribute definition has the following components:

node.

The node. prefix is optional and usually omitted, as shown in the example.

type

The type governs whether the attribute can be overridden. OpsWorks Stacks attributes typically use one of the following types:

- `default` is the most commonly used type, because it allows the attribute to be overridden.
- `normal` defines an attribute that overrides one of the standard OpsWorks Stacks attribute values.

> **Note**
>
> Chef supports additional types, which aren't necessary for OpsWorks Stacks but might be useful for your project. For more information, see About Attributes.

attribute name

The attribute name uses the standard Chef node syntax, `[:`*`attribute`*`][:`*`subattribute`*`][...]`. You can use any names you like for your attributes. However, as discussed in Overriding Attributes, custom cookbook attributes are merged into the instance's node object, along with the attributes from the stack configuration and deployment attributes, and Chef's Ohai tool. Commonly used configuration names such as *port* or *user* might appear in a variety of cookbooks.

To avoid name collisions, the convention is to create qualified attribute names with at least two elements, as shown in the example. The first element should be unique and is typically based on a product name like *Apache*. It is followed by one or more subattributes that identify the particular value, such as `[:user]` or `[:port]`. You can use as many subattributes as are appropriate for your project.

value

An attribute can be set to the following types of values:

- A string, such as `default[:apache][:keepalive] = 'Off'`.
- A number (without quotes) such as `default[:apache][:timeout] = 120`.
- A Boolean value, which can be either `true` or `false` (no quotes).
- A list of values, such as `default[:apache][:listen_ports] = [ '80','443' ]`

The attribute file is a Ruby application, so you can also use node syntax and logical operators to assign values based on other attributes. For more information about how to define attributes, see About Attributes s. For examples of working attribute files, see the OpsWorks Stacks built-in cookbooks at https://github.com/aws/opsworks-cookbooks.

Templates

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You configure many packages by creating a configuration file and placing it in the appropriate directory. You can include a configuration file in your cookbook and copy it to the appropriate directory, but a more flexible approach is to have your recipes create the configuration file from a template. One advantage of a template is that you can use attributes to define the template's values. This allows you, for example, to modify a configuration file without touching the cookbook by using custom JSON to override the appropriate attribute values.

A template has essentially the same content and structure as the associated file. Here is an example file, `httpd.conf`.

```
ServerRoot "<%= node[:apache][:dir] %>"
<% if node[:platform] == "debian" || node[:platform] == "ubuntu" -%>
  LockFile /var/lock/apache2/accept.lock
<% else -%>
   LockFile logs/accept.lock
<% end -%>
PidFile <%= node[:apache][:pid_file] %>
Timeout <%= node[:apache][:timeout] %>
KeepAlive <%= node[:apache][:keepalive] %>
MaxKeepAliveRequests <%= node[:apache][:keepaliverequests] %>
KeepAliveTimeout <%= node[:apache][:keepalivetimeout] %>
<IfModule mpm_prefork_module>
    StartServers          <%= node[:apache][:prefork][:startservers] %>
```

```
    MinSpareServers       <%= node[:apache][:prefork][:minspareservers] %>
    MaxSpareServers       <%= node[:apache][:prefork][:maxspareservers] %>
    ServerLimit           <%= node[:apache][:prefork][:serverlimit] %>
    MaxClients            <%= node[:apache][:prefork][:maxclients] %>
    MaxRequestsPerChild   <%= node[:apache][:prefork][:maxrequestsperchild] %>
</IfModule>
...
```

The following example is the `httpd.conf` file that was generated for a Ubuntu instance:

```
ServerRoot "/etc/httpd"
LockFile logs/accept.lock
PidFile /var/run/httpd/httpd.pid
Timeout 120
KeepAlive Off
MaxKeepAliveRequests 100
KeepAliveTimeout 3
<IfModule mpm_prefork_module>
    StartServers          16
    MinSpareServers       16
    MaxSpareServers       32
    ServerLimit           400
    MaxClients            400
    MaxRequestsPerChild   10000
</IfModule>
...
```

Much of the template's text is simply copied from the template to the `httpd.conf` file. However, `<%= ... %>` content is handled as follows:

- Chef replaces `<%= node[:attribute][:sub_attribute][:...]%>` with the attribute's value.

  For example, `StartServers <%= node[:apache][:prefork][:startservers] %>` becomes `StartServers 16` in the `httpd.conf`.

- You can use `<%if-%>, <%else-%>, and <%end-%>` to conditionally select a value.

  The example sets a different file path for `accept.lock` depending on the platform.

> **Note**
>
> You are not limited to the attributes in your cookbook's attribute files. You can use any attribute in the instance's node object. For example, generated by a Chef tool called Ohai and also incorporated into the node object. For more information on attributes, see Overriding Attributes.

For more information on templates, including how to incorporate Ruby code, see About Templates.

Recipes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Recipes are Ruby applications that define a system's configuration. They install packages, create configuration files from templates, execute shell commands, create files and directories, and so on. You typically have OpsWorks Stacks execute recipes automatically when a lifecycle event occurs on the instance but you can also run them explicitly at any time by using the Execute Recipes stack command. For more information, see About Recipes.

A recipe typically consists largely of a series of *resources*, each of which represents the desired state of an aspect of the system. Each resource includes a set of attributes that define the desired state and specify what action is to be taken. Chef associates each resource with an appropriate *provider* that performs the action. For more information, see Resources and Providers Reference.

A `package` resource helps you manage software packages on Linux instances. The following example installs the Apache package.

```
...
package 'apache2' do
  case node[:platform]
```

```
  when 'centos','redhat','fedora','amazon'
    package_name 'httpd'
  when 'debian','ubuntu'
    package_name 'apache2'
  end
  action :install
end
...
```

Chef uses the appropriate package provider for the platform. Resource attributes are often just assigned a value, but you can use Ruby logical operations to perform conditional assignments. The example uses a `case` operator, which uses `node[:platform]` to identify the instance's operating system and sets the `package_name` attribute accordingly. You can insert attributes into a recipe by using the standard Chef node syntax and Chef replaces it with the associated value. You can use any attribute in the node object, not just your cookbook's attributes.

After determining the appropriate package name, the code segment ends with an `install` action, which installs the package. Other actions for this resource include `upgrade` and `remove`. For more information, see package.

It is often useful to break complex installation and configuration tasks into one or more subtasks, each implemented as a separate recipe, and have your primary recipe run them at the appropriate time. The following example shows the line of code that follows the preceding example:

```
include_recipe 'apache2::service'
```

To have a recipe execute a child recipe, use the `include_recipe` keyword, followed by the recipe name. Recipes are identified by using the standard Chef *`CookbookName`*`::`*`RecipeName`* syntax, where *`RecipeName`* omits the `.rb` extension.

> **Note**
>
> An `include_recipe` statement effectively executes the recipe at that point in the primary recipe. However, what actually happens is that Chef replaces each `include_recipe` statement with the specified recipe's code before it executes the primary recipe.

A `directory` resource represents a directory, such as the one that is to contain a package's files. The following `default.rb` resource creates a Linux log directory.

```
directory node[:apache][:log_dir] do
    mode 0755
    action :create
end
```

The log directory is defined in one of the cookbook's attribute files. The resource specifies the directory's mode as 0755, and uses a `create` action to create the directory. For more information, see directory. You can also use this resource with Windows instances.

The `execute` resource represents commands, such as shell commands or scripts. The following example generates module.load files.

```
execute 'generate-module-list' do
  if node[:kernel][:machine] == 'x86_64'
    libdir = 'lib64'
  else
    libdir = 'lib'
  end
  command "/usr/local/bin/apache2_module_conf_generate.pl /usr/#{libdir}/httpd/
modules /etc/httpd/mods-available"
  action :run
end
```

The resource first determines the CPU type. `[:kernel][:machine]` is another of the automatic attributes that Chef generates to represent various system properties, the CPU type in this case. It then specifies the command, a Perl script and uses a `run` action to run the script, which generates the module.load files. For more information, see execute.

A `template` resource represents a file—typically a configuration file—that is to be generated from one of the cookbook's template files. The following example creates an `httpd.conf` configuration file from the `apache2.conf.erb` template that was discussed in Templates.

```
template 'apache2.conf' do
  case node[:platform]
  when 'centos','redhat','fedora','amazon'
    path "#{node[:apache][:dir]}/conf/httpd.conf"
  when 'debian','ubuntu'
```

```
    path "#{node[:apache][:dir]}/apache2.conf"
  end
  source 'apache2.conf.erb'
  owner 'root'
  group 'root'
  mode 0644
  notifies :restart, resources(:service => 'apache2')
end
```

The resource determines the generated file's name and location based on the instance's operating system. It then specifies `apache2.conf.erb` as the template to be used to generate the file and sets the file's owner, group, and mode. It runs the `notify` action to notify the `service` resource that represents the Apache server to restart the server. For more information, see template.

Stack Configuration and Deployment Attributes: Linux

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This topic includes the most commonly used stack configuration and deployment attributes and their associated node syntax. It is organized around the stack configuration namespace structure that is used by Linux stacks. Note that the same attribute names are sometimes used for different purposes, and occur in different namespaces. For example, `id` can refer to a stack ID, a layer ID, an app ID, and so on, so you need the fully qualified name to use the attribute value. A convenient way to visualize this data is as a JSON object. For examples , see Stack Configuration and Deployment Attributes.

Note

On Linux instances, OpsWorks Stacks installs this JSON object on each instance in addition to adding the data to the node object. You can retrieve it by using the agent CLI's `get_json` command.

Topics

- opsworks Attributes
- opsworks_custom_cookbooks Attributes
- dependencies Attributes
- ganglia Attributes
- mysql Attributes
- passenger Attributes
- opsworks_bundler Attributes
- deploy Attributes
- Other Top-Level Attributes

opsworks Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The `opsworks` element—sometimes referred to as the `opsworks` namespace—contains a set of attributes that define the basic stack configuration.

> **Important**
>
> Overriding the attribute values in the `opsworks` namespace is not recommended. Doing so can cause the built-in recipes to fail.

Topics

- applications
- instance Attributes
- layers Attributes

- rails_stack Attributes
- stack Attributes
- Other Top-level opsworks Attributes

applications

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Contains a list of embedded objects, one for each app that exists for the stack. Each embedded object contains the following attributes that describe the application configuration.

> **Note**
>
> The general node syntax for these attributes is as follows, where *i* specifies the instance's zero-based list index.
>
> ```
> node["opsworks"]["applications"]["i"]["attribute_name"]
> ```

application_type

The application's type (string). Possible values are as follows:

- php: PHP app
- `rails`: A Ruby on Rails app
- `java`: A Java app
- `nodejs`: A Node.js app
- `web`: A static HTML page
- `other`: All other application types

```
node["opsworks"]["applications"]["i"]["application_type"]
```

name

The user-defined display name, such as `"SimplePHP"` (string).

```
node["opsworks"]["applications"]["i"]["name"]
```

slug_name

A short name , which is an all-lowercase name such as `"simplephp"` that is generated by OpsWorks from the app's name (string).

```
node["opsworks"]["applications"]["i"]["slug_name"]
```

instance Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The `instance` attribute contains a set of attributes that specify the configuration of this instance.

architecture	availability_zone	backends
aws_instance_id	hostname	id
instance_type	ip	layers
private_dns_name	private_ip	public_dns_name
region		

architecture

The instance's architecture, such as "i386" (string).

```
node["opsworks"]["instance"]["architecture"]
```

availability_zone

The instance's availability zone, such as "us-west-2a" (string).

```
node["opsworks"]["instance"]["availability_zone"]
```

backends

The number of back-end web processes (string). It determines, for example, the number of concurrent connections that HAProxy will forward to a Rails back end. The default value depends on the instance's memory and number of cores.

```
node["opsworks"]["instance"]["backends"]
```

aws_instance_id

The EC2 instance ID (string).

```
node["opsworks"]["instance"]["aws_instance_id"]
```

hostname

The host name, such as "php-app1" (string).

```
node["opsworks"]["instance"]["hostname"]
```

id

The instance ID, which is an OpsWorks Stacks-generated GUID that uniquely identifies the instance (string).

```
node["opsworks"]["instance"]["id"]
```

instance_type

The instance type, such as "c1.medium" (string).

```
node["opsworks"]["instance"]["instance_type"]
```

ip

The public IP address (string).

```
node["opsworks"]["instance"]["ip"]
```

layers

A list of the instance's layers, which are identified by their short names, such as "lb" or "db-master" (list of string).

```
node["opsworks"]["instance"]["layers"]
```

private_dns_name

The private DNS name (string).

```
node["opsworks"]["instance"]["private_dns_name"]
```

private_ip

The private IP address (string).

```
node["opsworks"]["instance"]["private_ip"]
```

public_dns_name

The public DNS name (string).

```
node["opsworks"]["instance"]["public_dns_name"]
```

region

The AWS region, such as "us-west-2" (string).

```
node["opsworks"]["instance"]["region"]
```

layers Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The `layers` attribute contains a set of layer attributes, one for each of the stack's layers, which are named with the layer's short name, such as `php-app`. A stack can have at most one each of the built-in layers, whose short names are as follows:

- `db-master`: MySQL layer
- `java-app`: Java App Server layer
- `lb`: HAProxy layer
- `monitoring-master`: Ganglia layer
- `memcached`: Memcached layer
- `nodejs-app`: Node.js App Server layer
- `php-app`: PHP App Server layer
- `rails-app`: Rails App Server layer
- `web`: Static Web Server layer

A stack can contain any number of custom layers, which have user-defined short names.

Each layer attribute contains the following attributes:

- id
- instances
- name

id

The layer ID, which is a GUID that is generated by OpsWorks and uniquely identifies the layer (string).

```
node["opsworks"]["layers"]["layershortname"]["id"]
```

instances

The `instances` element contains a set of instance attributes, one for each of the layer's online instances. They are named with the instance's host name, such as `php-app1`.

> **Note**
>
> The `instances` element contains only those instances that are in the online state when the particular stack configuration and deployment attributes are created.

Each instance element contains the following attributes:

availability_zone	aws_instance_id	backends
booted_at	created_at	elastic_ip
instance_type	ip	private_ip
public_dns_name	private_dns_name	region
status		

availability_zone

The Availability Zone, such as `"us-west-2a"` (string).

```
node["opsworks"]["layers"]["layershortname"]["instances"]["instancehostname"]
["availability_zone"]
```

aws_instance_id

The EC2 instance ID (string).

```
node["opsworks"]["layers"]["layershortname"]["instances"]["instancehostname"]
["aws_instance_id"]
```

backends

The number of back-end web processes (number). It determines, for example, the number of concurrent connections that HAProxy will forward to a Rails back end. The default value depends on the instance's memory and number of cores.

```
node["opsworks"]["layers"]["layershortname"]["instances"]["instancehostname"]
["backends"]
```

booted_at

The time that the EC2 instance was booted, using the UTC yyyy-mm-dddThh:mm:ss+hh:mm format (string). For example, `"2013-10-01T08:35:22+00:00"` corresponds to 8:35:22 on Oct. 10, 2013, with no time zone offset. For more information, see ISO 8601.

```
node["opsworks"]["layers"]["layershortname"]["instances"]["instancehostname"]
["booted_at"]
```

created_at

The time that the EC2 instance was created, using the UTC yyyy-mm-dddThh:mm:ss+hh:mm format (string). For example, `"2013-10-01T08:35:22+00:00"` corresponds to 8:35:22 on Oct. 10, 2013, with no time zone offset. For more information, see ISO 8601.

```
node["opsworks"]["layers"]["layershortname"]["instances"]["instancehostname"]
["created_at"]
```

elastic_ip

The Elastic IP address, which is set to null if the instance does not have one (string).

```
node["opsworks"]["layers"]["layershortname"]["instances"]["instancehostname"]
["elastic_ip"]
```

instance_type

The instance type, such as `"c1.medium"` (string).

```
node["opsworks"]["layers"]["layershortname"]["instances"]["instancehostname"]
["instance_type"]
```

ip

The public IP address (string).

```
node["opsworks"]["layers"]["layershortname"]["instances"]["instancehostname"]
["ip"]
```

private_ip

The private IP address (string).

```
node["opsworks"]["layers"]["layershortname"]["instances"]["instancehostname"]
["private_ip"]
```

public_dns_name

The public DNS name (string).

```
node["opsworks"]["layers"]["layershortname"]["instances"]["instancehostname"]
["public_dns_name"]
```

private_dns_name

The private DNS name (string).

```
node["opsworks"]["layers"]["layershortname"]["instances"]["instancehostname"]
["private_dns_name"]
```

region

The AWS region, such as `"us-west-2"` (string).

```
node["opsworks"]["layers"]["layershortname"]["instances"]["instancehostname"]
["region"]
```

status

The status (string). Possible values are as follows:

- "requested"
- "booting"
- "running_setup"
- "online"
- "setup_failed"
- "start_failed"
- "terminating"
- "terminated"
- "stopped"
- "connection_lost"

```
node["opsworks"]["layers"]["layershortname"]["instances"]["instancehostname"]
["status"]
```

name

The layer's name, which is used to represent the layer in the console (string). It can be user-defined and is not necessarily unique.

```
node["opsworks"]["layers"]["layershortname"]["name"]
```

rails_stack Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

name

Specifies the rails stack, and is set to "apache_passenger" or "nginx_unicorn" (string).

```
node["opsworks"]["rails_stack"]["name"]
```

recipe

The associated recipe, which depends on whether you are using Passenger or Unicorn (string):

- Unicorn: `"unicorn::rails"`
- Passenger: `"passenger_apache2::rails"`

```
node["opsworks"]["rails_stack"]["recipe"]
```

restart_command

The restart command, which depends on whether you are using Passenger or Unicorn (string):

- Unicorn: `"../../shared/scripts/unicorn clean-restart"`
- Passenger: `"touch tmp/restart.txt"`

service

The service name, which depends on whether you are using Passenger or Unicorn (string):

- Unicorn: `"unicorn"`
- Passenger: `"apache2"`

```
node["opsworks"]["rails_stack"]["service"]
```

stack Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

`stack` attributes specify some aspects of the stack configuration, such as service layer configurations.

- elb-load-balancers
- id
- name
- rds_instances
- vpc_id

elb-load-balancers

Contains a list of embedded objects, one for each Elastic Load Balancing load balancer in the stack. Each embedded object contains the following attributes that describe the load balancer configuration.

> **Note**
>
> The general node syntax for these attributes is as follows, where *i* specifies the instance's zero-based list index.
>
> ```
> node["opsworks"]["stack"]["elb-load-balancers"]["i"]["attribute_name"]
> ```

dns_name

The load balancer's DNS name (string).

```
node["opsworks"]["stack"]["elb-load-balancers"]["i"]["dns_name"]
```

name

The load balancer's name (string).

```
node["opsworks"]["stack"]["elb-load-balancers"]["i"]["name"]
```

layer_id

The ID of the layer that the load balancer is attached to (string).

```
node["opsworks"]["stack"]["elb-load-balancers"]["i"]["layer_id"]
```

id

The stack ID (string).

```
node["opsworks"]["stack"]["id"]
```

name

The stack name (string).

```
node["opsworks"]["stack"]["name"]
```

rds_instances

Contains a list of embedded objects, one for each Amazon RDS instance that is registered with the stack. Each embedded object contains a set of attributes that define the instance's configuration. You specify these values when you use the Amazon RDS console or API to create the instance. You can also use the Amazon RDS console or API to edit some of the settings after the instance has been created. For more information, see the Amazon RDS documentation.

> **Note**
>
> The general node syntax for these attributes is as follows, where *i* specifies the instance's zero-based list index.
>
> ```
> node["opsworks"]["stack"]["rds_instances"]["i"]["attribute_name"]
> ```

If your stack has multiple Amazon RDS instances, the following is an example of how to use a particular instance in a recipe.

```
if my_rds = node["opsworks"]["stack"]["rds_instances"].select{|rds_instance|
 rds_instance["db_instance_identifier"] == 'db_id' }.first
  template "/etc/rds.conf" do
    source "rds.conf.erb"
    variables :address => my_rds["address"]
  end
end
```

address	allocated_storage	arn
auto_minor_version_upgrade	availability_zone	backup_retention_period
db_instance_class	db_instance_identifier	db_instance_status
db_name	db_parameter_groups	db_security_groups
db_user	engine	instance_create_time
license_model	multi_az	option_group_memberships
port	preferred_backup_window	preferred_maintenance_window
publicly_accessible	read_replica_db_instance_identifiers	region
status_infos	vpc_security_groups	

address

The instances URL, such as `opsinstance.ccdvt3hwog1a.us-west-2.rds.amazonaws.com` (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["address"]
```

allocated_storage

The allocated storage, in GB (number).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["allocated_storage"]
```

arn

The instance's ARN (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["arn"]
```

auto_minor_version_upgrade

Whether to automatically apply minor version upgrades (Boolean).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["auto_minor_version_upgrade"]
```

availability_zone

The instance's Availability Zone, such as `us-west-2a` (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["availability_zone"]
```

backup_retention_period

The backup retention period, in days (number).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["backup_retention_period"]
```

db_instance_class

The DB instance class, such as `db.m1.small` (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["db_instance_class"]
```

db_instance_identifier

The user-defined DB instance identifier (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["db_instance_identifier"]
```

db_instance_status

The instance's status (string). For more information, see DB Instance.

```
node["opsworks"]["stack"]["rds_instances"]["i"]["db_instance_status"]
```

db_name

The user-defined DB name (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["db_name"]
```

db_parameter_groups

The instance's DB parameter groups, which contains a list of embedded objects, one for each parameter group. For more information, see Working with DB Parameter Groups. Each object contains the following attributes:

db_parameter_group_name

The group name (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["db_parameter_groups"][j"]
["db_parameter_group_name"]
```

parameter_apply_status

The apply status (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["db_parameter_groups"][j"]
["parameter_apply_status"]
```

db_security_groups

The instance's database security groups, which contains a list of embedded objects, one for each security group. For more information, see Working with DB Security Groups. Each object contains the following attributes

db_security_group_name

The security group name (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["db_security_groups"][j"]
["db_security_group_name"]
```

status

The status (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["db_security_groups"][j"]
["status"]
```

db_user

The user-defined Master User name (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["db_user"]
```

engine

The database engine, such as `mysql(5.6.13)` (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["engine"]
```

instance_create_time

The instance creation time, such as `2014-04-15T16:13:34Z` (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["instance_create_time"]
```

license_model

The instance's license model, such as `general-public-license` (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["license_model"]
```

multi_az

Whether multi-AZ deployment is enabled (Boolean).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["multi_az"]
```

option_group_memberships

The instance's option group memberships, which contains a list of embedded objects, one for each option group. For more information, see Working with Option Groups. Each object contains the following attributes:

option_group_name

The group's name (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["option_group_memberships"]
[j"]["option_group_name"]
```

status

The group's status (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["option_group_memberships"]
[j"]["status"]
```

port

The database server's port (number).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["port"]
```

preferred_backup_window

The preferred daily backup window, such as `06:26-06:56` (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["preferred_backup_window"]
```

preferred_maintenance_window

The preferred weekly maintenance window, such as `thu:07:13-thu:07:43` (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["preferred_maintenance_window"]
```

publicly_accessible

Whether the database is publicly accessible (Boolean).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["publicly_accessible"]
```

read_replica_db_instance_identifiers

A list of the read-replica instance identifiers (list of string). For more information, see Working with Read Replicas.

```
node["opsworks"]["stack"]["rds_instances"]["i"]
["read_replica_db_instance_identifiers"]
```

region

The AWS region, such as `us-west-2` (string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["region"]
```

status_infos

A list of status information (list of string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["status_infos"]
```

vpc_security_groups

A list of VPC security groups (list of string).

```
node["opsworks"]["stack"]["rds_instances"]["i"]["vpc_security_groups"]
```

vpc_id

The VPC id (string). This value is `null` if the instance is not in a VPC.

```
node["opsworks"]["stack"]["vpc_id"]
```

Other Top-level opsworks Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section contains the `opsworks` attributes that do not have child attributes.

activity

The activity that is associated with the attributes, such as `deploy` (string).

```
node["opsworks"]["activity"]
```

agent_version

The version of the instance's OpsWorks agent (string).

```
node["opsworks"]["agent_version"]
```

deploy_chef_provider

The Chef deploy provider, which influences a deployed app's directory structure (string). You can set this attribute to one the following:

- `Branch`
- `Revision`
- `Timestamped` (default value)

```
node["opsworks"]["deploy_chef_provider"]
```

ruby_stack

The Ruby stack (string). The default setting is the enterprise version (`ruby_enterprise`). For the MRI version, set this attribute to `ruby`.

```
node["opsworks"]["ruby_stack"]
```

ruby_version

The Ruby version that will be used by applications (string). You can use this attribute to specify only the major and minor version. You must use the appropriate ["ruby"] attribute to specify the patch version. For more information about how to specify a version, including examples, see Ruby Versions. For complete details on how OpsWorks Stacks determines the Ruby version, see the built-in attributes file, ruby.rb.

```
node["opsworks"]["ruby_version"]
```

run_cookbook_tests

Whether to run minitest-chef-handler tests on your Chef 11.4 cookbooks (Boolean).

```
node["opsworks"]["run_cookbook_tests"]
```

sent_at

When this command was sent to the instance (number).

```
node["opsworks"]["sent_at"]
```

deployment

If these attributes are associated with a deploy activity, `deployment` is set to the deployment ID, an OpsWorks Stacks-generated GUID that uniquely identifies the deployment (string). Otherwise the attribute is set to null.

```
node["opsworks"]["deployment"]
```

opsworks_custom_cookbooks Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Contains attributes that specify the stack's custom cookbooks.

enabled

Whether custom cookbooks are enabled (Boolean).

```
node["opsworks_custom_cookbooks"]["enabled"]
```

recipes

A list of the recipes that are to be executed for this command, including custom recipes, using the *cookbookname*::*recipename* format (list of string).

```
node["opsworks_custom_cookbooks"]["recipes"]
```

dependencies Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Contains several attributes that are related to the `update_dependencies` stack command.

gem_binary

The location of the Gems binary (string).

upgrade_debs

Whether to upgrade Debs packages (Boolean).

update_debs

Whether to update Debs packages (Boolean).

ganglia Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Contains a **web** attribute that contains several attributes that specify how to access the Ganglia statistics web page:

password

The password required to access the statistics page (string).

```
node["ganglia"]["web"]["password"]
```

url

The statistics page's URL path, such as `"/ganglia"` (string). The complete URL is `http://`*`DNSNameURLPath`*, where *`DNSName`* is the associated instance's DNS name.

```
node["ganglia"]["web"]["url"]
```

user

The user name required to access the statistics page (string).

```
node["ganglia"]["web"]["user"]
```

mysql Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Contains a set of attributes that specify the MySQL database server configuration.

clients

A list of client IP addresses (list of string).

```
node["mysql"]["clients"]
```

server_root_password

The root password (string).

```
node["mysql"]["server_root_password"]
```

passenger Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Contains a set of attributes that specify the Phusion Passenger configuration.

gem_bin

The location of the RubyGems binaries, such as "`/usr/local/bin/gem`" (string).

```
node["passenger"]["gem_bin"]
```

max_pool_size

The maximum pool size (number).

```
node["passenger"]["max_pool_size"]
```

ruby_bin

The location of the Ruby binaries, such as "`/usr/local/bin/ruby`".

```
node["passenger"]["ruby_bin"]
```

version

The Passenger version (string).

```
node["passenger"]["version"]
```

opsworks_bundler Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Contains elements that specify Bundler support.

manage_package

Whether to install and manage Bundler (Boolean).

```
node["opsworks_bundler"]["manage_package"]
```

version

The bundler version (string).

```
node["opsworks_bundler"]["version"]
```

deploy Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

If the attributes are associated with a Deploy event or an Execute Recipes stack command, the `deploy` attribute contains an attribute for each app that was deployed, named by the app's short name. Each app attribute contains the following attributes:

application	application_type	auto_bundle_on_deploy
database	deploy_to	domains
document_root	environment_variables	group
keep_releases	memcached	migrate
mounted_at	purge_before_symlink	rails_env
restart_command	scm	ssl_certificate
ssl_certificate_ca	ssl_certificate_key	ssl_support
stack	symlink_before_migrate	symlinks
user		

application

The app's slug name, such as `"simplephp"` (string).

```
node["deploy"]["appshortname"]["application"]
```

application_type

The app type (string). Possible values are as follows:

- `java`: A Java app
- `nodejs`: A Node.js app
- `php`: A PHP app
- `rails`: A Ruby on Rails app
- `web`: A static HTML page
- `other`: All other application types

```
node["deploy"]["appshortname"]["application_type"]
```

auto_bundle_on_deploy

For Rails applications, whether to execute bundler during the deployment (Boolean).

```
node["deploy"]["appshortname"]["auto_bundle_on_deploy"]
```

database

Contains the information required to connect the app's database. If the app has an attached a database layer, OpsWorks Stacks automatically assigns the appropriate values to these attributes.

adapter

The database adapter, such as `mysql` (string).

```
node["deploy"]["appshortname"]["database"]["adapter"]
```

database

The database name, which is usually the app's slug name, such as `"simplephp"` (string).

```
node["deploy"]["appshortname"]["database"]["database"]
```

data_source_provider

The data source: `mysql` or `rds` (string).

```
node["deploy"]["appshortname"]["database"]["data_source_provider"]
```

host

The database host's IP address (string).

```
node["deploy"]["appshortname"]["database"]["host"]
```

password

The database password (string).

```
node["deploy"]["appshortname"]["database"]["password"]
```

port

The database port (number).

```
node["deploy"]["appshortname"]["database"]["port"]
```

reconnect

For Rails applications, whether the application should reconnect if the connection no longer exists (Boolean).

```
node["deploy"]["appshortname"]["database"]["reconnect"]
```

username

The user name (string).

```
node["deploy"]["appshortname"]["database"]["username"]
```

deploy_to

Where the app is to be deployed to, such as `"/srv/www/simplephp"` (string).

```
node["deploy"]["appshortname"]["deploy_to"]
```

domains

A list of the app's domains (list of string).

```
node["deploy"]["appshortname"]["domains"]
```

document_root

The document root, if you specify a nondefault root, or null if you use the default root (string).

```
node["deploy"]["appshortname"]["document_root"]
```

environment_variables

A collection of up to twenty attributes that represent the user-specified environment variables that have been defined for the app. For more information about how to define an app's environment variables, see Adding Apps. Each attribute name is set to an environment variable name and the corresponding value is set to the variable's value, so you can use the following syntax to reference a particular value.

```
node["deploy"]["appshortname"]["environment_variables"]["variable_name"]
```

group

The app's group (string).

```
node["deploy"]["appshortname"]["group"]
```

keep_releases

The number of app deployments that OpsWorks Stacks will store (number). This attribute controls the number of times you can roll back an app. By default, it is set to the global value, deploy_keep_releases , which has a default value of 5. You can override `keep_releases` to specify the number of stored deployments for a particular application.

```
node["deploy"]["appshortname"]["keep_releases"]
```

memcached

Contains two attributes that define the memcached configuration.

host

The Memcached server instance's IP address (string).

```
node["deploy"]["appshortname"]["memcached"]["host"]
```

port

The port that the memcached server is listening on (number).

```
node["deploy"]["appshortname"]["memcached"]["port"]
```

migrate

For Rails applications, whether to run migrations (Boolean).

```
node["deploy"]["appshortname"]["migrate"]
```

mounted_at

The app's mount point, if you specify a nondefault mount point, or null if you use the default mount point (string).

```
node["deploy"]["appshortname"]["mounted_at"]
```

purge_before_symlink

For Rails apps, an array of paths to be cleared before creating symlinks (list of string).

```
node["deploy"]["appshortname"]["purge_before_symlink"]
```

rails_env

For Rails App Server instances, the rails environment, such as `"production"` (string).

```
node["deploy"]["appshortname"]["rails_env"]
```

restart_command

A command to be run when the app is restarted, such as `"echo 'restarting app'"`.

```
node["deploy"]["appshortname"]["restart_command"]
```

scm

Contains a set of attributes that specify the information that OpsWorks uses to deploy the app from its source control repository. The attributes vary depending on the repository type.

password

The password, for private repositories, and null for public repositories (string). For private Amazon S3 buckets, the attribute is set to the secret key.

```
node["deploy"]["appshortname"]["scm"]["password"]
```

repository

The repository URL, such as `"git://github.com/amazonwebservices/opsworks-demo-php-simple-app.git"` (string).

```
node["deploy"]["appshortname"]["scm"]["repository"]
```

revision

If the repository has multiple branches, the attribute specifies the app's branch or version, such as `"version1"` (string). Otherwise it is set to null.

```
node["deploy"]["appshortname"]["scm"]["revision"]
```

scm_type

The repository type (string). Possible values are as follows:

- `"git"`: A Git repository
- `"svn"`: A Subversion repository
- `"s3"`: An Amazon S3 bucket
- `"archive"`: An HTTP archive
- `"other"`: Another repository type

```
node["deploy"]["appshortname"]["scm"]["scm_type"]
```

ssh_key

A deploy SSH key, for accessing private Git repositories, and null for public repositories (string).

```
node["deploy"]["appshortname"]["scm"]["ssh_key"]
```

user

The user name, for private repositories, and null for public repositories (string). For private Amazon S3 buckets, the attribute is set to the access key.

```
node["deploy"]["appshortname"]["scm"]["user"]
```

ssl_certificate

The app's SSL certificate, if you enabled SSL support, or null otherwise (string).

```
node["deploy"]["appshortname"]["ssl_certificate"]
```

ssl_certificate_ca

If SSL is enabled, an attribute for specifying an intermediate certificate authority key or client authentication (string).

```
node["deploy"]["appshortname"]["ssl_certificate_ca"]
```

ssl_certificate_key

The app's SSL private key, if you enabled SSL support, or null otherwise (string).

```
node["deploy"]["appshortname"]["ssl_certificate_key"]
```

ssl_support

Whether SSL is supported (Boolean).

```
node["deploy"]["appshortname"]["ssl_support"]
```

stack

Contains one Boolean attribute, `needs_reload`, that specifies whether to reload the app server during deployment.

```
node["deploy"]["appshortname"]["stack"]["needs_reload"]
```

symlink_before_migrate

For Rails apps, contains symlinks that are to be created before running migrations as "*link*":"*target*" pairs.

```
node["deploy"]["appshortname"]["symlink_before_migrate"]
```

symlinks

Contains the deployment's symlinks as "*link*":"*target*" pairs.

```
node["deploy"]["appshortname"]["symlinks"]
```

user

The app's user (string).

```
node["deploy"]["appshortname"]["user"]
```

Other Top-Level Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section contains top-level stack configuration attributes that do not have child attributes.

rails Attributes

Contains a **max_pool_size** attribute that specifies the server's maximum pool size (number). The attribute value is set by OpsWorks Stacks and depends on the instance type, but you can override it by using custom JSON or a custom attribute file.

```
node["rails"]["max_pool_size"]
```

recipes Attributes

A list of the built-in recipes that were run by this activity, using the "*cookbookname*::*recipename*" format (list of string).

```
node["recipes"]
```

opsworks_rubygems Attributes

Contains a **version** element that specifies the RubyGems version (string).

```
node["opsworks_rubygems"]["version"]
```

languages Attributes

Contains an attribute for each installed language, named for the language, such as **ruby**. The attribute is an object that contains an attribute, such as **ruby_bin**, that specifies the installation folder, such as "`/usr/bin/ruby`" (string).

ssh_users Attributes

Contains a set of attributes, each of which describes one of the users that have been granted SSH permissions. Each attribute is named with a user's Unix ID. OpsWorks Stacks generates a unique ID for each user in the 2000-4000 range, such as "`2001`", and creates a user with that ID on every instance. Because OpsWorks reserves the 2000-4000 range, users that you create outside of OpsWorks (by using cookbook recipes, or by importing users into OpsWorks from IAM, for example) can have UIDs that are overwritten by OpsWorks Stacks for another user. As a best practice, create users and manage their access in the OpsWorks Stacks console. If you do create users outside of OpsWorks Stacks, use *`UnixID`* values greater than 4000.

Each attribute contains the following attributes:

email

The user's e-mail address (string).

```
node["ssh_users"]["UnixID"]["email"]
```

public_key

The user's public SSH key (string).

```
node["ssh_users"]["UnixID"]["public_key"]
```

sudoer

Whether the user has sudo permissions (Boolean).

```
node["ssh_users"]["UnixID"]["sudoer"]
```

name

The user name (string).

```
node["ssh_users"]["UnixID"]["name"]
```

Built-in Cookbook Attributes

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

Most of these attributes are available only on Linux stacks.

Most of the built-in recipes have one or more attributes files that define various settings. You can access these settings in your custom recipes and use custom JSON to override them. You typically need to access or override attributes that control the configuration of the various server technologies that are supported by OpsWorks Stacks. This section summarizes those attributes. The complete attributes files, and the associated recipes and templates, are available at https://github.com/aws/opsworks-cookbooks.git.

Note

All built-in recipe attributes are `default` type.

Topics

- apache2 Attributes
- deploy Attributes
- haproxy Attributes
- memcached Attributes
- mysql Attributes
- nginx Attributes
- opsworks_berkshelf Attributes
- opsworks_java Attributes
- passenger_apache2 Attributes
- ruby Attributes
- unicorn Attributes

apache2 Attributes

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

These attributes are available only on Linux stacks.

The apache2 attributes specify the Apache HTTP server configuration. For more information, see Apache Core Features. For more information on how to override built-in attributes to specify custom values, see Overriding Attributes.

binary	contact	deflate_types

dir	document_root	group
hide_info_headers	icondir	init_script
keepalive	keepaliverequests	keepalivetimeout
lib_dir	libexecdir	listen_ports
log_dir	logrotate Attributes	pid_file
prefork Attributes	serversignature	servertokens
timeout	traceenable	user
version	worker Attributes	

binary

The location of the Apache binary (string). The default value is `'/usr/sbin/httpd'`.

```
node[:apache][:binary]
```

contact

An e-mail contact (string). The default value is a dummy address, `'ops@example.com'`.

```
node[:apache][:contact]
```

deflate_types

Directs `mod_deflate` to enable compression for the specified Mime types, if they are supported by the browser (list of string). The default value is as follows:

```
['application/javascript',
 'application/json',
 'application/x-javascript',
 'application/xhtml+xml',
 'application/xml',
 'application/xml+rss',
 'text/css',
```

```
'text/html',
'text/javascript',
'text/plain',
'text/xml']
```

> **Warning**
>
> Compression can introduce security risks. To completely disable compression, set this attribute as follows:
>
> ```
> node[:apache][:deflate_types] = []
> ```

```
node[:apache][:deflate_types]
```

dir

The server's root directory (string). The default values are as follows:

- Amazon Linux and Red Hat Enterprise Linux (RHEL): `'/etc/httpd'`
- Ubuntu: `'/etc/apache2'`

```
node[:apache][:dir]
```

document_root

The document root (string). The default values are as follows:

- Amazon Linux and RHEL: `'/var/www/html'`
- Ubuntu: `'/var/www'`

```
node[:apache][:document_root]
```

group

The group name (string). The default values are as follows:

- Amazon Linux and RHEL: `'apache'`
- Ubuntu: `'www-data'`

```
node[:apache][:group]
```

hide_info_headers

Whether to omit version and module information from HTTP headers (`'true'/'false'`) (string). The default value is `'true'`.

```
node[:apache][:hide_info_headers]
```

icondir

The icon directory (string). The defaults value are as follows:

- Amazon Linux and RHEL: `'/var/www/icons/'`
- Ubuntu: `'/usr/share/apache2/icons'`

```
node[:apache][:icondir]
```

init_script

The initialization script (string). The default values are as follows:

- Amazon Linux and RHEL: `'/etc/init.d/httpd'`
- Ubuntu: `'/etc/init.d/apache2'`

```
node[:apache][:init_script]
```

keepalive

Whether to enable keep-alive connections (string). The possible values are `'On'` and `'Off'` (string). The default value is `'Off'`.

```
node[:apache][:keepalive]
```

keepaliverequests

The maximum number of keep-alive requests that Apache will handle at the same time (number). The default value is `100`.

```
node[:apache][:keepaliverequests]
```

keepalivetimeout

The time that Apache waits for a request before closing the connection (number). The default value is 3.

```
node[:apache][:keepalivetimeout]
```

lib_dir

The directory that contains the object code libraries (string). The default values are as follows:

- Amazon Linux (x86): `'/usr/lib/httpd'`
- Amazon Linux (x64) and RHEL: `'/usr/lib64/httpd'`
- Ubuntu: `'/usr/lib/apache2'`

```
node[:apache][:lib_dir]
```

libexecdir

The directory that contains the program executables (string). The default values are as follows:

- Amazon Linux (x86): `'/usr/lib/httpd/modules'`
- Amazon Linux (x64) and RHEL: `'/usr/lib64/httpd/modules'`
- Ubuntu: `'/usr/lib/apache2/modules'`

```
node[:apache][:libexecdir]
```

listen_ports

A list of ports that the server listens to (list of string). The default value is `[ '80','443' ]`.

```
node[:apache][:listen_ports]
```

log_dir

The log directory (string). The default values are as follows:

- Amazon Linux and RHEL: `'/var/log/httpd'`
- Ubuntu: `'/var/log/apache2'`

```
node[:apache][:log_dir]
```

logrotate Attributes

These attributes specify how to rotate the log files.

delaycompress

Whether to delay compressing a closed log file until the start of the next rotation cycle (`'true'`/`'false'`) (string). The default value is `'true'`.

```
node[:apache][:logrotate][:delaycompress]
```

group

The log files' group (string). The default value is `'adm'`.

```
node[:apache][:logrotate][:group]
```

mode

The log files' mode (string). The default value is `'640'`.

```
node[:apache][:logrotate][:mode]
```

owner

The log files' owner (string). The default value is `'root'`.

```
node[:apache][:logrotate][:owner]
```

rotate

The number of rotation cycles before a closed log file is removed (string). The default value is `'30'`.

```
node[:apache][:logrotate][:rotate]
```

schedule

The rotation schedule (string). Possible values are as follows:

- `'daily'`

- `'weekly'`
- `'monthly'`

The default value is `'daily'`.

```
node[:apache][:logrotate][:schedule]
```

pid_file

The file that contains the daemon's process ID (string). The default values are as follows:

- Amazon Linux and RHEL: `'/var/run/httpd/httpd.pid'`
- Ubuntu: `'/var/run/apache2.pid'`

```
node[:apache][:pid_file]
```

prefork Attributes

These attributes specify the pre-forking configuration.

maxclients

The maximum number of simultaneous requests that will be served (number). The default value is `400`.

> **Note**
>
> Use this attribute only for instances that are running Amazon Linux, or RHEL. If your instances are running Ubuntu 14.04 LTS, use maxrequestworkers.

```
node[:apache][:prefork][:maxclients]
```

maxrequestsperchild

The maximum number of requests that a child server process will handle (number). The default value is `10000`.

```
node[:apache][:prefork][:maxrequestsperchild]
```

maxrequestworkers

The maximum number of simultaneous requests that will be served (number). The default value is `400`.

> **Note**
>
> Use this attribute only for instances that are running Ubuntu 14.04 LTS. If your instances are running Amazon Linux, or RHEL, use maxclients .

```
node[:apache][:prefork][:maxrequestworkers]
```

maxspareservers

The maximum number of idle child server processes (number). The default value is 32.

```
node[:apache][:prefork][:maxspareservers]
```

minspareservers

The minimum number of idle child server processes (number). The default value is 16.

```
node[:apache][:prefork][:minspareservers]
```

serverlimit

The maximum number of processes that can be configured (number). The default value is `400`.

```
node[:apache][:prefork][:serverlimit]
```

startservers

The number of child server processes to be created at startup (number). The default value is 16.

```
node[:apache][:prefork][:startservers]
```

serversignature

Specifies whether and how to configure a trailing footer for server-generated documents (string). The possible values are `'On'`, `'Off'`, and `'Email'`). The default value is `'Off'`.

```
node[:apache][:serversignature]
```

servertokens

Specifies what type of server version information is included in the response header (string):

- `'Full'`: Full information. For example, Server: Apache/2.4.2 (Unix) PHP/4.2.2 MyMod/1.2
- `'Prod'`: Product name. For example, Server: Apache
- `'Major'`: Major version. For example, Server: Apache/2
- `'Minor'`: Major and minor version. For example, Server: Apache/2.4
- `'Min'`: Minimal version. For example, Server: Apache/2.4.2
- `'OS'`: Version with operating system. For example, Server: Apache/2.4.2 (Unix)

The default value is `'Prod'`.

```
node[:apache][:servertokens]
```

timeout

The amount of time that Apache waits for I/O (number). The default value is `120`.

```
node[:apache][:timeout]
```

traceenable

Whether to enable TRACE requests (string). The possible values are `'On'` and `'Off'`. The default value is `'Off'`.

```
node[:apache][:traceenable]
```

user

The user name (string). The default values are as follows:

- Amazon Linux and RHEL: `'apache'`

- Ubuntu: `'www-data'`

```
node[:apache][:user]
```

version

The Apache version (string). The default values are as follows:

- Amazon Linux: `2.2`
- Ubuntu 14.04 LTS: `2.4`
- RHEL: `2.4`

```
node[:apache][:version]
```

worker Attributes

These attributes specify the worker process configuration.

startservers

The number of child server processes to be created at startup (number). The default value is `4`.

```
node[:apache][:worker][:startservers]
```

maxclients

The maximum number of simultaneous requests that will be served (number). The default value is `1024`.

```
node[:apache][:worker][:maxclients]
```

maxsparethreads

The maximum number of idle threads (number). The default value is `192`.

```
node[:apache][:worker][:maxsparethreads]
```

minsparethreads

The minimum number of idle threads (number). The default value is `64`.

```
node[:apache][:worker][:minsparethreads]
```

threadsperchild

The number of threads per child process (number). The default value is 64.

```
node[:apache][:worker][:threadsperchild]
```

maxrequestsperchild

The maximum number of requests that a child server process will handle (number). The default value is `10000`.

```
node[:apache][:worker][:maxrequestsperchild]
```

deploy Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The built-in deploy cookbook's `deploy.rb` attributes file defines the following attributes in the `opsworks` namespace. For more information on deploy directories, see Deploy Recipes. For more information on how to override built-in attributes to specify custom values, see Overriding Attributes.

deploy_keep_releases

A global setting for the number of app deployments that OpsWorks Stacks will store (number). The default value is 5. This value controls the number of times you can roll back an app.

```
node[:opsworks][:deploy_keep_releases]
```

group

(Linux only) The `group` setting for the app's deploy directory (string). The default value depends on the instance's operating system:

- For Ubuntu instances, the default value is `www-data`.
- For Amazon Linux or RHEL instances that are members of a Rails App Server layer that uses Nginx and Unicorn, the default value is `nginx`.
- For all other Amazon Linux or RHEL instances, the default value is `apache`.

```
node[:opsworks][:deploy_user][:group]
```

user

(Linux only) The `user` setting for the app's deploy directory (string). The default value is `deploy`.

```
node[:opsworks][:deploy_user][:user]
```

haproxy Attributes

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

These attributes are available only on Linux stacks.

The `haproxy attributes` specify the HAProxy server configuration. For more information, see HAProxy Docs. For more information on how to override built-in attributes to specify custom values, see Overriding Attributes.

balance	check_interval	client_timeout
connect_timeout	default_max_connections	global_max_connections
health_check_method	health_check_url	queue_timeout
http_request_timeout	maxcon_factor_nodejs_app	maxcon_factor_nodejs_app_ssl
maxcon_factor_php_app	maxcon_factor_php_app_ssl	maxcon_factor_rails_app
maxcon_factor_rails_app_ssl	maxcon_factor_static	maxcon_factor_static_ssl
retries	server_timeout	stats_url
stats_user		

balance

The algorithm used by a load balancer to select a server (string). The default value is `'roundrobin'`. The other options are:

- 'static-rr'
- 'leastconn'
- 'source'
- 'uri'
- 'url_param'
- 'hdr(name)'
- 'rdp-cookie'
- 'rdp-cookie(name)'

For more information on these arguments, see balance.

```
node[:haproxy][:balance]
```

check_interval

The health check time interval (string). The default value is `'10s'`.

```
node[:haproxy][:check_interval]
```

client_timeout

The maximum amount of time that a client can be inactive (string). The default value is `'60s'`.

```
node[:haproxy][:client_timeout]
```

connect_timeout

The maximum amount of time that HAProxy will wait for a server connection attempt to succeed (string). The default value is `'10s'`.

```
node[:haproxy][:connect_timeout]
```

default_max_connections

The default maximum number of connections (string). The default value is `'80000'`.

```
node[:haproxy][:default_max_connections]
```

global_max_connections

The maximum number of connections (string). The default value is `'80000'`.

```
node[:haproxy][:global_max_connections]
```

health_check_method

The health check method (string). The default value is `'OPTIONS'`.

```
node[:haproxy][:health_check_method]
```

health_check_url

The URL path that is used to check servers' health (string). The default value is `'/'`.

```
node[:haproxy][:health_check_url ]
```

queue_timeout

The maximum wait time for a free connection (string). The default value is `'120s'`.

```
node[:haproxy][:queue_timeout]
```

http_request_timeout

The maximum amount of time that HAProxy will wait for a complete HTTP request (string). The default value is `'30s'`.

```
node[:haproxy][:http_request_timeout]
```

retries

The number of retries after server connection failure (string). The default value is `'3'`.

```
node[:haproxy][:retries]
```

server_timeout

The maximum amount of time that a client can be inactive (string). The default value is `'60s'`.

```
node[:haproxy][:server_timeout]
```

stats_url

The URL path for the statistics page (string). The default value is `'/haproxy?stats'`.

```
node[:haproxy][:stats_url]
```

stats_user

The statistics page user name (string). The default value is `'opsworks'`.

```
node[:haproxy][:stats_user]
```

The `maxcon` attributes represent a load factor multiplier that is used to compute the maximum number of connections that HAProxy allows for backends. For example, suppose you have a Rails app server on a small instance with a `backend` value of 4, which means that OpsWorks Stacks will configure four Rails processes for that instance. If you use the default `maxcon_factor_rails_app` value of 7, HAProxy will handle 28 (4*7) connections to the Rails server.

maxcon_factor_nodejs_app

The maxcon factor for a Node.js app server (number). The default value is 10.

```
node[:haproxy][:maxcon_factor_nodejs_app]
```

maxcon_factor_nodejs_app_ssl

The maxcon factor for a Node.js app server with SSL (number). The default value is 10.

```
node[:haproxy][:maxcon_factor_nodejs_app_ssl]
```

maxcon_factor_php_app

The maxcon factor for a PHP app server (number). The default value is 10.

```
node[:haproxy][:maxcon_factor_php_app]
```

maxcon_factor_php_app_ssl

The maxcon factor for a PHP app server with SSL (number). The default value is 10.

```
node[:haproxy][:maxcon_factor_php_app_ssl]
```

maxcon_factor_rails_app

The maxcon factor for a Rails app server (number). The default value is 7.

```
node[:haproxy][:maxcon_factor_rails_app]
```

maxcon_factor_rails_app_ssl

The maxcon factor for a Rails app server with SSL (number). The default value is 7.

```
node[:haproxy][:maxcon_factor_rails_app_ssl]
```

maxcon_factor_static

The maxcon factor for a static web server (number). The default value is 15.

```
node[:haproxy][:maxcon_factor_static]
```

maxcon_factor_static_ssl

The maxcon factor for a static web server with SSL (number). The default value is 15.

```
node[:haproxy][:maxcon_factor_static_ssl]
```

memcached Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> These attributes are available only on Linux stacks.

The `memcached attributes` specify the Memcached server configuration. For more information on how to override built-in attributes to specify custom values, see Overriding Attributes.

memory	max_connections	pid_file
port	start_command	stop_command
user		

memory

The maximum memory to use, in MB (number). The default value is 512.

```
node[:memcached][:memory]
```

max_connections

The maximum number of connections (string). The default value is `'4096'`.

```
node[:memcached][:max_connections]
```

pid_file

The file that contains the daemon's process ID (string). The default value is `'var/run/memcached.pid'`.

```
node[:memcached][:pid_file]
```

port

The port to listen on (number). The default value is `11211`.

```
node[:memcached][:port]
```

start_command

The start command (string). The default value is `'/etc/init.d/memcached start'`.

```
node[:memcached][:start_command]
```

stop_command

The stop command (string). The default value is `'/etc/init.d/memcached stop'`.

```
node[:memcached][:stop_command]
```

user

The user (string). The default value is `'nobody'`.

```
node[:memcached][:user]
```

mysql Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> These attributes are available only on Linux stacks.

The `mysql` attributes specify the MySQL master configuration. For more information, see Server System Variables. For more information on how to override built-in attributes to specify custom values, see Overriding Attributes.

basedir	bind_address	clients
conf_dir	confd_dir	datadir
grants_path	mysql_bin	mysqladmin_bin
pid_file	port	root_group
server_root_password	socket	tunable Attributes

basedir

The base directory (string). The default value is `'/usr'`.

```
node[:mysql][:basedir]
```

bind_address

The address that MySQL listens on (string). The default value is `'0.0.0.0'`.

```
node[:mysql][:bind_address]
```

clients

A list of clients (list of string).

```
node[:mysql][:clients]
```

conf_dir

The directory that contains the configuration file (string). The default values are as follows:

- Amazon Linux and RHEL: `'/etc'`
- Ubuntu: `'/etc/mysql'`

```
node[:mysql][:conf_dir]
```

confd_dir

The directory that contains additional configuration files (string). The default value is `'/etc/mysql/conf.d'`.

```
node[:mysql][:confd_dir]
```

datadir

The data directory (string). The default value is `'/var/lib/mysql'`.

```
node[:mysql][:datadir]
```

grants_path

The grant table location (string). The default value is `'/etc/mysql_grants.sql'`.

```
node[:mysql][:grants_path]
```

mysql_bin

The mysql binaries location (string). The default value is `'/usr/bin/mysql'`.

```
node[:mysql][:mysql_bin]
```

mysqladmin_bin

The mysqladmin location (string). The default value is `'/usr/bin/mysqladmin'`.

```
node[:mysql][:mysqladmin_bin]
```

pid_file

The file that contains the daemon's process ID (string). The default value is `'/var/run/mysqld/mysqld.pid'`.

```
node[:mysql][:pid_file]
```

port

The port that the server listens on (number). The default value is `3306`.

```
node[:mysql][:port]
```

root_group

The root group (string). The default value is `'root'`.

```
node[:mysql][:root_group]
```

server_root_password

The server's root password (string). The default value is randomly generated.

```
node[:mysql][:server_root_password]
```

socket

The location of the socket file (string). The default value is `'/var/lib/mysql/mysql.sock'`. The default values are as follows:

- Amazon Linux and RHEL: `'/var/lib/mysql/mysql.sock'`
- Ubuntu: `'/var/run/mysqld/mysqld.sock'`

```
node[:mysql][:socket]
```

tunable Attributes

The tunable attributes are used for performance tuning.

back_log	innodb_additional_mem_pool_size	innodb_buffer_pool_size
innodb_flush_log_at_trx_commit	innodb_lock_wait_timeout	key_buffer
log_slow_queries	long_query_time	max_allowed_packet
max_connections	max_heap_table_size	net_read_timeout
net_write_timeout	query_cache_limit	query_cache_size
query_cache_type	thread_cache_size	thread_stack
wait_timeout	table_cache	

back_log

The maximum number of outstanding requests (string). The default value is `'128'`.

```
node[:mysql][:tunable][:back_log]
```

innodb_additional_mem_pool_size

The size of the pool that Innodb uses to store internal data structures (string). The default value is `'20M'`.

```
node[:mysql][:tunable][:innodb_additional_mem_pool_size]
```

innodb_buffer_pool_size

The Innodb buffer pool size (string). The attribute value is set by OpsWorks Stacks and depends on the instance type, but you can override it by using custom JSON or a custom attribute file.

```
node[:mysql][:tunable][:innodb_buffer_pool_size]
```

innodb_flush_log_at_trx_commit

How often Innodb flushes the log buffer (string). The default value is `'2'`. For more information, see innodb_flush_log_at_trx_commit.

```
node[:mysql][:tunable][:innodb_flush_log_at_trx_commit]
```

innodb_lock_wait_timeout

The maximum amount of time, in seconds, that an Innodb transaction waits for a row lock (string). The default value is `'50'`.

```
node[:mysql][:tunable][:innodb_lock_wait_timeout]
```

key_buffer

The index buffer size (string). The default value is `'250M'`.

```
node[:mysql][:tunable][:key_buffer]
```

log_slow_queries

The location of the slow-query log file (string). The default value is `'/var/log/mysql/mysql-slow.log'`.

```
node[:mysql][:tunable][:log_slow_queries]
```

long_query_time

The time, in seconds, required to designate a query as a long query (string). The default value is `'1'`.

```
node[:mysql][:tunable][:long_query_time]
```

max_allowed_packet

The maximum allowed packet size (string). The default value is `'32M'`.

```
node[:mysql][:tunable][:max_allowed_packet]
```

max_connections

The maximum number of concurrent client connections (string). The default value is `'2048'`.

```
node[:mysql][:tunable][:max_connections]
```

max_heap_table_size

The maximum size of user-created `MEMORY` tables (string). The default value is `'32M'`.

```
node[:mysql][:tunable][:max_heap_table_size]
```

net_read_timeout

The amount of time, in seconds, to wait for more data from a connection (string). The default value is `'30'`.

```
node[:mysql][:tunable][:net_read_timeout]
```

net_write_timeout

The amount of time, in seconds, to wait for a block to be written to a connection (string). The default value is `'30'`.

```
node[:mysql][:tunable][:net_write_timeout]
```

query_cache_limit

The maximum size of an individual cached query (string). The default value is `'2M'`.

```
node[:mysql][:tunable][:query_cache_limit]
```

query_cache_size

The query cache size (string). The default value is `'128M'`.

```
node[:mysql][:tunable][:query_cache_size]
```

query_cache_type

The query cache type (string). The possible values are as follows:

- `'0'`: No caching or retrieval of cached data.
- `'1'`: Cache statements that don't begin with `SELECT SQL_NO_CACHE`.
- `'2'`: Cache statements that begin with `SELECT SQL_CACHE`.

The default value is `'1'`.

```
node[:mysql][:tunable][:query_cache_type]
```

thread_cache_size

The number of client threads that are cached for re-use (string). The default value is `'8'`.

```
node[:mysql][:tunable][:thread_cache_size]
```

thread_stack

The stack size for each thread (string). The default value is `'192K'`.

```
node[:mysql][:tunable][:thread_stack]
```

wait_timeout

The amount of time, in seconds, to wait on a noninteractive connection. The default value is `'180'` (string).

```
node[:mysql][:tunable][:wait_timeout]
```

table_cache

The number of open tables (string). The default value is `'2048'`.

```
node[:mysql][:tunable][:table_cache]
```

nginx Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> These attributes are available only on Linux stacks.

The `nginx` attributes specify the Nginx configuration. For more information, see Directive Index. For more information on how to override built-in attributes to specify custom values, see Overriding Attributes.

binary	dir	gzip
gzip_comp_level	gzip_disable	gzip_http_version
gzip_proxied	gzip_static	gzip_types
gzip_vary	keepalive	keepalive_timeout
log_dir	user	server_names_hash_bucket_size
worker_processes	worker_connections	

binary

The location of the Nginx binaries (string). The default value is `'/usr/sbin/nginx'`.

```
node[:nginx][:binary]
```

dir

The location of files such as configuration files (string). The default value is `'/etc/nginx'`.

```
node[:nginx][:dir]
```

gzip

Whether gzip compression is enabled (string). The possible values are `'on'` and `'off'`. The default value is `'on'`.

> **Warning**
>
> Compression can introduce security risks. To completely disable compression, set this attribute as follows:
>
> ```
> node[:nginx][:gzip] = 'off'
> ```

```
node[:nginx][:gzip]
```

gzip_comp_level

The compression level, which can range from 1–9, with 1 corresponding to the least compression (string). The default value is `'2'`.

```
node[:nginx][:gzip_comp_level]
```

gzip_disable

Disables gzip compression for specified user agents (string). The value is a regular expression and the default value is `'MSIE [1-6].(?!.*SV1)'`.

```
node[:nginx][:gzip_disable]
```

gzip_http_version

Enables gzip compression for a specified HTTP version (string). The default value is `'1.0'`.

```
node[:nginx][:gzip_http_version]
```

gzip_proxied

Whether and how to compress the response to proxy requests, which can take one of the following values (string):

- `'off'`: do not compress proxied requests
- `'expired'`: compress if the Expire header prevents caching
- `'no-cache'`: compress if the Cache-Control header is set to "no-cache"
- `'no-store'`: compress if the Cache-Control header is set to "no-store"
- `'private'`: compress if the Cache-Control header is set to "private"
- `'no_last_modified'`: compress if Last-Modified is not set
- `'no_etag'`: compress if the request lacks an ETag header
- `'auth'`: compress if the request includes an Authorization header
- `'any'`: compress all proxied requests

The default value is `'any'`.

```
node[:nginx][:gzip_proxied]
```

gzip_static

Whether the gzip static module is enabled (string). The possible values are `'on'` and `'off'`. The default value is `'on'`.

```
node[:nginx][:gzip_static]
```

gzip_types

A list of MIME types to be compressed (list of string). The default value is `['text/plain', 'text/html', 'text/css', 'application/x-javascript', 'text/xml', 'application/xml', 'application/xml+rss', 'text/javascript']`.

```
node[:nginx][:gzip_types]
```

gzip_vary

Whether to enable a `Vary:Accept-Encoding` response header (string). The possible values are `'on'` and `'off'`. The default value is `'on'`.

```
node[:nginx][:gzip_vary]
```

keepalive

Whether to enable a keep-alive connection (string). The possible values are `'on'` and `'off'`. The default value is `'on'`.

```
node[:nginx][:keepalive]
```

keepalive_timeout

The maximum amount of time, in seconds, that a keep-alive connection remains open (number). The default value is 65.

```
node[:nginx][:keepalive_timeout]
```

log_dir

The location of the log files (string). The default value is `'/var/log/nginx'`.

```
node[:nginx][:log_dir]
```

user

The user (string). The default values are as follows:

- Amazon Linux and RHEL: `'www-data'`
- Ubuntu: `'nginx'`

```
node[:nginx][:user]
```

server_names_hash_bucket_size

The bucket size for hash tables of server names, which can be set to 32, 64, or 128 (number). The default value is 64.

```
node[:nginx][:server_names_hash_bucket_size]
```

worker_processes

The number of worker processes (number). The default value is `10`.

```
node[:nginx][:worker_processes]
```

worker_connections

The maximum number of worker connections (number). The default value is `1024`. The maximum number of clients is set to `worker_processes * worker_connections`.

```
node[:nginx][:worker_connections]
```

opsworks_berkshelf Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> These attributes are available only on Linux stacks.

The `opsworks_berkshelf` attributes specify the Berkshelf configuration. For more information, see Berkshelf. For more information on how to override built-in attributes to specify custom values, see Overriding Attributes.

debug

Whether to include Berkshelf debugging information in the Chef log (Boolean). The default value is `false`.

```
node['opsworks_berkshelf]['debug']
```

opsworks_java Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> These attributes are available only on Linux stacks.

The `opsworks_java attributes` specify the Tomcat server configuration. For more information, see Apache Tomcat Configuration Reference. For more information on how to override built-in attributes to specify custom values, see Overriding Attributes.

datasources	java_app_server_version	java_shared_lib_dir
jvm_pkg Attributes	custom_pkg_location_url_debian	java_home_basedir
custom_pkg_location_url_rhel	use_custom_pkg_location	jvm_options
jvm_version	tomcat Attributes	

datasources

A set of attributes that define JNDI resource names (string). For more information on how to use this attribute, see Deploying a JSP App with a Back-End Database. The default value is an empty hash, which can be filled with custom mappings between app short names and JNDI names. For more information, see Deploying a JSP App with a Back-End Database.

```
node['opsworks_java']['datasources']
```

java_app_server_version

The Java app server version (number). The default value is 7. You can override this attribute to specify version 6. If you install a nondefault JDK, this attribute is ignored.

```
node['opsworks_java']['java_app_server_version']
```

java_shared_lib_dir

The directory for the Java shared libraries (string). The default value is `/usr/share/java`.

```
node['opsworks_java']['java_shared_lib_dir']
```

jvm_pkg Attributes

A set of attributes that you can override to install a nondefault JDK.

use_custom_pkg_location

Whether to install a custom JDK instead of OpenJDK (Boolean). The default value is `false`.

```
node['opsworks_java']['jvm_pkg']['use_custom_pkg_location']
```

custom_pkg_location_url_debian

The location of the JDK package to be installed on Ubuntu instances (string). The default value is `'http://aws.amazon.com/'`, which is simply an initialization value with no proper meaning. If you want to install a nondefault JDK, you must override this attribute and set it to the appropriate URL.

```
node['opsworks_java']['jvm_pkg']['custom_pkg_location_url_debian']
```

custom_pkg_location_url_rhel

The location of the JDK package to be installed on Amazon Linux and RHEL instances (string). The default value is `'http://aws.amazon.com/'`, which is simply an initialization value with no proper meaning. If you want to install a nondefault JDK, you must override this attribute and set it to the appropriate URL.

```
node['opsworks_java']['jvm_pkg']['custom_pkg_location_url_rhel']
```

java_home_basedir

The directory that the JDK package will be extracted to (string). The default value is `/usr/local`. You do not need to specify this setting for RPM packages; they include a complete directory structure.

```
node['opsworks_java']['jvm_pkg']['java_home_basedir']
```

jvm_options

The JVM command line options, which allow you to specify settings such as the heap size (string). A common set of options is `-Djava.awt.headless=true -Xmx128m -XX:+UseConcMarkSweepGC`. The default value is no options.

```
node['opsworks_java']['jvm_options']
```

jvm_version

The OpenJDK version (number). The default value is 7. You can override this attribute to specify OpenJDK version 6. If you install a nondefault JDK, this attribute is ignored.

```
node['opsworks_java']['jvm_version']
```

tomcat Attributes

A set of attributes that you can override to install the default Tomcat configuration.

ajp_port	apache_tomcat_bind_mod	apache_tomcat_bind_path
auto_deploy	connection_timeout	mysql_connector_jar
port	secure_port	shutdown_port
threadpool_max_threads	threadpool_min_spare_threads	unpack_wars
uri_encoding	use_ssl_connector	use_threadpool

userdatabase_pathname		

ajp_port

The AJP port (number). The default value is `8009`.

```
node['opsworks_java']['tomcat]['ajp_port']
```

apache_tomcat_bind_mod

The proxy module (string). The default value is `proxy_http`. You can override this attribute to specify the AJP proxy module, `proxy_ajp`.

```
node['opsworks_java']['tomcat]['apache_tomcat_bind_mod']
```

apache_tomcat_bind_path

The Apache-Tomcat bind path (string). The default value is `/`. You should not override this attribute; changing the bind path can cause the application to stop working.

```
node['opsworks_java']['tomcat]['apache_tomcat_bind_path']
```

auto_deploy

Whether to autodeploy (Boolean). The default value is `true`.

```
node['opsworks_java']['tomcat]['auto_deploy']
```

connection_timeout

The connection timeout, in milliseconds (number). The default value is `20000` (20 seconds).

```
node['opsworks_java']['tomcat]['connection_timeout']
```

mysql_connector_jar

The MySQL connector library's JAR file (string). The default value is `mysql-connector-java.jar`.

```
node['opsworks_java']['tomcat]['mysql_connector_jar']
```

port

The standard port (number). The default value is `8080`.

```
node['opsworks_java']['tomcat]['port']
```

secure_port

The secure port (number). The default value is `8443`.

```
node['opsworks_java']['tomcat]['secure_port']
```

shutdown_port

The shutdown port (number). The default value is `8005`.

```
node['opsworks_java']['tomcat]['shutdown_port']
```

threadpool_max_threads

The maximum number of threads in the thread pool (number). The default value is `150`.

```
node['opsworks_java']['tomcat]['threadpool_max_threads']
```

threadpool_min_spare_threads

The minimum number of spare threads in the thread pool (number). The default value is `4`.

```
node['opsworks_java']['tomcat]['threadpool_min_spare_threads']
```

unpack_wars

Whether to unpack WAR files (Boolean). The default value is `true`.

```
node['opsworks_java']['tomcat]['unpack_wars']
```

uri_encoding

The URI encoding (string). The default value is `UTF-8`.

```
node['opsworks_java']['tomcat]['uri_encoding']
```

use_ssl_connector

Whether to use an SSL connector (Boolean). The default value is `false`.

```
node['opsworks_java']['tomcat]['use_ssl_connector']
```

use_threadpool

Whether to use a thread pool (Boolean). The default value is `false`.

```
node['opsworks_java']['tomcat]['use_threadpool']
```

userdatabase_pathname

The user database path name (string). The default value is `conf/tomcat-users.xml`.

```
node['opsworks_java']['tomcat]['userdatabase_pathname']
```

passenger_apache2 Attributes

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

These attributes are available only on Linux stacks.

The passenger_apache2 attributes specify the Phusion Passenger configuration. For more information, see Phusion Passenger users guide, Apache version. For more information on how to override built-in attributes to specify custom values, see Overriding Attributes.

friendly_error_pages	gem_bin	gems_path
high_performance_mode	root_path	max_instances_per_app
max_pool_size	max_requests	module_path
pool_idle_time	rails_app_spawner_idle_time	rails_framework_spawner_idle_time
rails_spawn_method	ruby_bin	ruby_wrapper_bin
stat_throttle_rate	version	

friendly_error_pages

Whether to display a friendly error page if an application fails to start (string). This attribute can be set to 'on' or 'off'; the default value is 'off'.

```
node[:passenger][:friendly_error_pages]
```

gem_bin

The location of the Gem binaries (string). The default value is `'/usr/local/bin/gem'`.

```
node[:passenger][:gem_bin]
```

gems_path

The gems path (string). The default value depends on the Ruby version. For example:

- Ruby version 1.8: `'/usr/local/lib/ruby/gems/1.8/gems'`
- Ruby version 1.9: `'/usr/local/lib/ruby/gems/1.9.1/gems'`

```
node[:passenger][:gems_path]
```

high_performance_mode

Whether to use Passenger's high-performance mode (string). The possible values are `'on'` and `'off'`. The default value is `'off'`.

```
node[:passenger][:high_performance_mode ]
```

root_path

The Passenger root directory (string). The default value depends on the Ruby and Passenger versions. In Chef syntax, the value is `"#{node[:passenger][:gems_path]}/passenger-#{passenger[:version]}"`.

```
node[:passenger][:root_path]
```

max_instances_per_app

The maximum number of application processes per app (number). The default value is `0`. For more information, see PassengerMaxInstancesPerApp.

```
node[:passenger][:max_instances_per_app]
```

max_pool_size

The maximum number of application processors (number). The default value is 8. For more information, see PassengerMaxPoolSize.

```
node[:passenger][:max_pool_size]
```

max_requests

The maximum number of requests (number). The default value is `0`.

```
node[:passenger][:max_requests]
```

module_path

The module path (string). The default values are as follows:

- Amazon Linux and RHEL: `"#{node['apache']['libexecdir']}/mod_passenger.so"`
- Ubuntu: `"#{passenger[:root_path]}/ext/apache2/mod_passenger.so"`

```
node[:passenger][:module_path]
```

pool_idle_time

The maximum time, in seconds, that an application process can be idle (number). The default value is `14400` (4 hours). For more information, see PassengerPoolIdleTime.

```
node[:passenger][:pool_idle_time]
```

rails_app_spawner_idle_time

The maximum idle time for the Rails app spawner (number). If this attribute is set to zero, the app spawner does not time out. The default value is `0`. For more information, see Spawning Methods Explained.

```
node[:passenger][:rails_app_spawner_idle_time]
```

rails_framework_spawner_idle_time

The maximum idle time for the Rails framework spawner (number). If this attribute is set to zero, the framework spawner does not time out. The default value is `0`. For more information, see Spawning Methods Explained.

```
node[:passenger][:rails_framework_spawner_idle_time]
```

rails_spawn_method

The Rails spawn method (string). The default value is `'smart-lv2'`. For more information, see Spawning Methods Explained.

```
node[:passenger][:rails_spawn_method]
```

ruby_bin

The location of the Ruby binaries (string). The default value is `'/usr/local/bin/ruby'`.

```
node[:passenger][:ruby_bin]
```

ruby_wrapper_bin

The location of the Ruby wrapper script (string). The default value is `'/usr/local/bin/ruby_gc_wrapper.sh'`.

```
node[:passenger][:ruby_wrapper_bin]
```

stat_throttle_rate

The rate at which Passenger performs file system checks (number). The default value is 5, which means that the checks will be performed at most once every 5 seconds. For more information, see PassengerStatThrottleRate .

```
node[:passenger][:stat_throttle_rate]
```

version

The version (string). The default value is `'3.0.9'`.

```
node[:passenger][:version]
```

ruby Attributes

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

These attributes are available only on Linux stacks.

The `ruby attributes` specify the Ruby version that is used by applications. Note that attribute usage changed with the introduction of semantic versioning in Ruby 2.1. For more information about how to specify a version, including examples, see Ruby Versions. For complete details on how OpsWorks Stacks determines the Ruby version, see the built-in attributes file, ruby.rb. For more information on how to override built-in attributes to specify custom values, see Overriding Attributes.

full_version

The full version number (string). You should not override this attribute. Instead, use [:opsworks][:ruby_version] and the appropriate patch version attribute to specify a version.

```
[:ruby][:full_version]
```

major_version

The major version number (string). You should not override this attribute. Instead, use [:opsworks][:ruby_version] to specify the major version.

```
[:ruby][:major_version]
```

minor_version

The minor version number (string). You should not override this attribute. Instead, use [:opsworks][:ruby_version] to specify the minor version.

```
[:ruby][:minor_version]
```

patch

The patch level (string). This attribute is valid for Ruby version 2.0.0 and earlier. For later Ruby versions, use the `patch_version` attribute.

```
[:ruby][:patch]
```

The patch number must be prefaced by p. For example, you would use the following custom JSON to specify patch level 484.

```
{
```

```
  "ruby":{"patch":"p484"}
}
```

patch_version

The patch number (string). This attribute is valid for Ruby version 2.1 and later. For earlier Ruby versions, use the `patch` attribute.

```
[:ruby][:patch_version]
```

pkgrelease

The package release number (string).

```
[:ruby][:pkgrelease]
```

unicorn Attributes

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

These attributes are available only on Linux stacks.

The `unicorn` attributes specify the Unicorn configuration. For more information, see Unicorn::Configurator. For more information on how to override built-in attributes to specify custom values, see Overriding Attributes.

accept_filter	backlog	delay

tcp_nodelay	tcp_nopush	preload_app
timeout	tries	version
worker_processes		

accept_filter

The accept filter, `'httpready'` or `'dataready'` (string). The default value is `'httpready'`.

```
node[:unicorn][:accept_filter]
```

backlog

The maximum number of requests that the queue can hold (number). The default value is `1024`.

```
node[:unicorn][:backlog]
```

delay

The amount of time, in seconds, to wait to retry binding a socket (number). The default value is `0.5`.

```
node[:unicorn][:delay]
```

preload_app

Whether to preload an app before forking a worker process (Boolean). The default value is `true`.

```
node[:unicorn][:preload_app]
```

tcp_nodelay

Whether to disable Nagle's algorithm for TCP sockets (Boolean). The default value is `true`.

```
node[:unicorn][:tcp_nodelay]
```

tcp_nopush

Whether to enable TCP_CORK (Boolean). The default value is `false`.

```
node[:unicorn][:tcp_nopush]
```

timeout

The maximum amount time, in seconds, that a worker is allowed to use for each request (number). Workers that exceed the timeout value are terminated. The default value is `60`.

```
node[:unicorn][:timeout]
```

tries

The maximum number of times to retry binding to a socket (number). The default value is 5.

```
node[:unicorn][:tries]
```

version

The Unicorn version (string). The default value is `'4.7.0'`.

```
node[:unicorn][:version]
```

worker_processes

The number of worker processes (number). The default value is `max_pool_size`, if it exists, and 4 otherwise.

```
node[:unicorn][:worker_processes]
```

Troubleshooting Chef 11.10 and Earlier Versions for Linux

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate

their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

For additional troubleshooting information, see Debugging and Troubleshooting Guide.

Chef Logs for Chef 11.10 and Earlier Versions for Linux

OpsWorks Stacks stores each instance's Chef logs in its `/var/lib/aws/opsworks/chef` directory. You need sudo privileges to access this directory. The log for each run is in a file named *YYYY*`-`*MM*`-`*DD*`-`*HH*`-`*MM*`-`*SS*`-`*NN*`.log`.

For more information, see the following:

- Viewing a Chef Log with the Console
- Viewing a Chef Log with the CLI or API
- Interpreting a Chef Log
- Common Chef Log Errors

Using OpsWorks Stacks with Other AWS Services

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can have application servers running in an OpsWorks Stacks stack use a variety of AWS services that are not directly integrated with OpsWorks Stacks. For example, you can have your

application servers use Amazon RDS as a back-end database. You can access such services by using the following general pattern:

1. Create and configure the AWS service by using the AWS console, API, or CLI and record any required configuration data that the application will need to access the service, such as host name or port.
2. Create one or more custom recipes to configure the application so that it can access the service.

   The recipe obtains the configuration data from stack configuration and deployment JSON attributes that you define with custom JSON prior to running the recipes.
3. Assign the custom recipe to the Deploy lifecycle event on the application server layer.
4. Create a custom JSON object that assigns appropriate values to the configuration data attributes and add it to your stack configuration and deployment JSON.
5. Deploy the application to the stack.

   Deployment runs the custom recipes, which use the configuration data values that you defined in the custom JSON to configure the application so that it can access the service.

This section describes how to have OpsWorks Stacks application servers access a variety of AWS services. It assumes that you are already familiar with Chef cookbooks and how recipes can use stack and configuration JSON attributes to configure applications, typically by creating configuration files. If not, you should first read Cookbooks and Recipes and Customizing OpsWorks Stacks.

Topics

- Using a Back-end Data Store
- Using ElastiCache Redis as an In-Memory Key-Value Store
- Using an Amazon S3 Bucket
- Using AWS CodePipeline with OpsWorks Stacks

Using a Back-end Data Store

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate

their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Application server stacks commonly include a database server to provide a back-end data store. OpsWorks Stacks provides integrated support for MySQL servers through the MySQL layer and for several types of database servers through the Amazon Relational Database Service (Amazon RDS) layer. However, you can easily customize a stack to have the application servers use other database servers such as Amazon DynamoDB or MongoDB. This topic describes the basic procedure for connecting an application server to an AWS database server. It uses the stack and application from Getting Started with Chef 11 Linux Stacks to show how to manually connect a PHP application server to an RDS database. Although the example is based on a Linux stack, the basic principles also apply to Windows stacks. For an example of how to incorporate a MongoDB database server into a stack, see Deploying MongoDB with OpsWorks.

Note

This topic uses Amazon RDS as a convenient example. However, if you want to use an Amazon RDS database with your stack, it's much easier to use an Amazon RDS layer.

Topics

- How to Set up a Database Connection
- How to Connect an Application Server Instance to Amazon RDS

How to Set up a Database Connection

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You set up the connection between an application server and its back-end database by using a custom recipe. The recipe configures the application server as required, typically by creating a configuration file. The recipe gets the connection data such as the host and database name from a set of attributes in the stack configuration and deployment attributes that OpsWorks Stacks installs on every instance.

For example, Step 2 of Getting Started with Chef 11 Linux Stacks is based on a stack named MyStack with two layers, PHP App Server and MySQL, each with one instance. You deploy an app named SimplePHPApp to the PHP App Server instance that uses the database on the MySQL instance as a back-end data store. When you deploy the application, OpsWorks Stacks installs stack configuration and deployment attributes that contain the database connection information. The following example shows the database connection attributes, represented as JSON:

```
{
  ...
  "deploy": {
    "simplephpapp": {
      ...
      "database": {
        "reconnect": true,
        "password": null,
        "username": "root",
        "host": null,
        "database": "simplephpapp"
        ...
      },
      ...
    }
  }
}
```

The attribute values are supplied by OpsWorks Stacks, and are either generated or based on user provided information.

To allow SimplePHPApp to access the data store, you must set up the connection between the PHP application server and the MySQL database by assigning a custom recipe named `appsetup.rb` to the PHP App Server layer's Deploy lifecycle event. When you deploy SimplePHPApp, OpsWorks Stacks runs `appsetup.rb`, which creates a configuration file named `db-connect.php` that sets up the connection, as shown in the following excerpt.

```
node[:deploy].each do |app_name, deploy|
  ...
  template "#{deploy[:deploy_to]}/current/db-connect.php" do
    source "db-connect.php.erb"
    mode 0660
    group deploy[:group]

    if platform?("ubuntu")
      owner "www-data"
    elsif platform?("amazon")
      owner "apache"
    end

    variables(
      :host =>     (deploy[:database][:host] rescue nil),
      :user =>     (deploy[:database][:username] rescue nil),
      :password => (deploy[:database][:password] rescue nil),
      :db =>       (deploy[:database][:database] rescue nil),
      :table =>    (node[:phpapp][:dbtable] rescue nil)
    )
    ...
  end
end
```

The variables that characterize the connection—`host`, `user`, and so on—are set the corresponding values from the deploy JSON's `[:deploy][:app_name][:database]` attributes. For simplicity, the example assumes that you have already created a table named `urler`, so the table name is represented by `[:phpapp][:dbtable]` in the cookbook's attributes file.

This recipe can actually connect the PHP application server to any MySQL database server, not just members of a MySQL layer. To use a different MySQL server, you just have to set the `[:database]` attributes to values that are appropriate for your server, which you can do by using custom JSON. OpsWorks Stacks then incorporates those attributes and values into the stack configuration and deployment attributes and `appsetup.rb` uses them to create the template that sets up the connection. For more information on overriding stack configuration and deployment JSON, see Overriding Attributes.

How to Connect an Application Server Instance to Amazon RDS

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section describes how to customize MyStack from Getting Started with Chef 11 Linux Stacks to have the PHP application server connect to an RDS instance.

Topics

- Create an Amazon RDS MySQL Database
- Customize the Stack to Connect to the RDS Database

Create an Amazon RDS MySQL Database

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Now you're ready to create an RDS database for the example using the Amazon RDS console's Launch DB Instance Wizard. The following procedure is a brief summary of the essential details. For a detailed description of how to create a database, see Getting Started with Amazon RDS.

To create the Amazon RDS database

1. If this is your first time creating an RDS database, click **Get Started Now**. Otherwise, click **RDS Dashboard** in the navigation pane, and then click **Launch a DB Instance**.

2. Select the **MySQL Community Edition** as the DB instance.
3. For **Do you plan to use this database for production purposes?** select **No, this instance...**, which is sufficient for the example. For production use, you might want to select **Yes, use Multi-AZ Deployment...**. Click **Next Step**.
4. On the **Specify DB Details** page, specify the following settings:

   - **DB Instance Class**: **db.t2.micro**
   - **Multi-AZ Deployment**: **No**
   - **Allocated Storage**: **5** GB
   - **DB Instance Identifier**: **`rdsexample`**
   - **Master Username**: **`opsworksuser`**
   - **Master Password**: Specify a suitable password and record it for later use.

   Accept the default settings for the other options and click **Next Step**.
5. On the **Configure Advanced Settings** page, specify the following settings:

   - In the **Network & Security** section, for **VPC Security Group(s)**, select **phpsecgroup (VPC)**
   - In the **Database Options** section, for **Database Name**, type **`rdsexampledb`**
   - In the **Backup** section, set **Backup Retention Period** to **0** for the purposes of this walkthrough.

   Accept the default settings for the other options and click **Launch DB Instance**.
6. Choose **View Your DB Instances** to see the list of DB instances.
7. Select the **rdsexample** instance in the list and click the arrow to reveal the instance endpoint and other details. Record the endpoint for later use. It will be something like `rdsexample.c6c8mntzhgv0.us-west-2.rds.amazonaws.com:3306`. Just record the DNS name; you won't need the port number.
8. Use a tool such as MySQL Workbench to create a table named `urler` in the `rdsexampledb` database by using following SQL command:

```
CREATE TABLE urler(id INT UNSIGNED NOT NULL AUTO_INCREMENT,author VARCHAR(63) NOT
 NULL,message TEXT,PRIMARY KEY (id))
```

Customize the Stack to Connect to the RDS Database

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Once you have created an RDS instance to use as a back-end database for the PHP application server, you can customize MyStack from Getting Started with Chef 11 Linux Stacks.

To connect the PHP App Server to an RDS database

1. Open the OpsWorks Stacks console and create a stack with a PHP App Server layer that contains one instance and deploy SimplePHPApp, as described in Getting Started with Chef 11 Linux Stacks. This stack uses version1 of SimplePHPApp, which does not use a database connection.
2. Update the stack configuration to use the custom cookbooks that include the `appsetup.rb` recipe, and related template and attribute files.
   1. Set **Use custom Chef cookbooks** to **Yes**.
   2. Set **Repository type** to **Git** and **Repository URL** to `git://github.com/amazonwebservices/opsworks-example-cookbooks.git`.
3. Add the following to the stack's **Custom Chef JSON** box to assign the RDS connection data to the `[:database]` attributes that `appsetup.rb` uses to create the configuration file.

```
{
  "deploy": {
    "simplephpapp": {
      "database": {
        "username": "opsworksuser",
        "password": "your_password",
        "database": "rdsexampledb",
        "host": "rds_endpoint",
        "adapter": "mysql"
```

```
      }
    }
  }
}
```

Use the following attribute values:

- **username**: The master user name that you specified when you created the RDS instance.

  This example uses `opsworksuser`.

- **password**: The master password that you specified when you created the RDS instance.

  Fill in the password that you specified.

- **database**: The database that you created when you created the RDS instance.

  This example uses `rdsexampledb`.

- **host**: The RDS instance's endpoint, which you got from the RDS console when you created the instance in the previous section. Don't include the port number.

- **adapter**: The adapter.

  The RDS instance for this example uses MySQL, so **adapter** is set to `mysql`. Unlike the other attributes, **adapter** is not used by `appsetup.rb`. It is instead used by the PHP App Server layer's built-in Configure recipe to create a different configuration file.

4. Edit the SimplePHPApp configuration to specify a version of SimplePHPApp that uses a back-end database, as follows:

   - **Document root**: Set this option to `web`.
   - **Branch/Revision**: Set this option to `version2`.

   Leave the remaining options unchanged.

5. Edit the PHP App Server layer to set up the database connection by adding `phpapp::appsetup` to the layer's Deploy recipes.

6. Deploy the new SimplePHPApp version.

7. When SimplePHPApp is deployed, run the application by going to the **Instances** page and clicking the php-app1 instance's public IP address. You should see the following page in your browser, which allows you to enter text and store it in the database.



Note

If your stack has a MySQL layer, OpsWorks Stacks automatically assigns the corresponding connection data to the `[:database]` attributes. However, if you assign custom JSON to the stack that defines different `[:database]` values, they override the default values. Because the `[:deploy]` attributes are installed on every instance, any recipes that depend on the `[:database]` attributes will use the custom connection data, not the MySQL layer's data for the. If you want a particular application server layer to use the custom connection data, assign the custom JSON to the layer's Deploy event, and restrict that deployment to that layer. For more information on how to use deployment attributes, see Deploying Apps. For more information on overriding OpsWorks Stacks built-in attributes, see Overriding Attributes.

Using ElastiCache Redis as an In-Memory Key-Value Store

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate

their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

This topic is based on a Linux stack, but Windows stacks can also use Amazon ElastiCache (ElastiCache). For an example of how of how to use ElastiCache with a Windows instance, see ElastiCache as an ASP.NET Session Store.

You can often improve application server performance by using a caching server to provide an in-memory key-value store for small items of data such as strings. Amazon ElastiCache is an AWS service that makes it easy to provide caching support for your application server, using either the Memcached or Redis caching engines. OpsWorks Stacks provides built-in support for Memcached. However, if Redis better suits your requirements, you can customize your stack so that your application servers use ElastiCache Redis.

This topic walks you through basic process of providing ElastiCache Redis caching support for Linux stacks, using a Rails application server as an example. It assumes that you already have an appropriate Ruby on Rails application. For more information on ElastiCache, see What Is Amazon ElastiCache?.

Topics

- Step 1: Create an ElastiCache Redis Cluster
- Step 2: Set up a Rails Stack
- Step 3: Create and Deploy a Custom Cookbook
- Step 4: Assign the Recipe to a LifeCycle Event
- Step 5: Add Access Information to the Stack Configuration JSON
- Step 6: Deploy and run the App

Step 1: Create an ElastiCache Redis Cluster

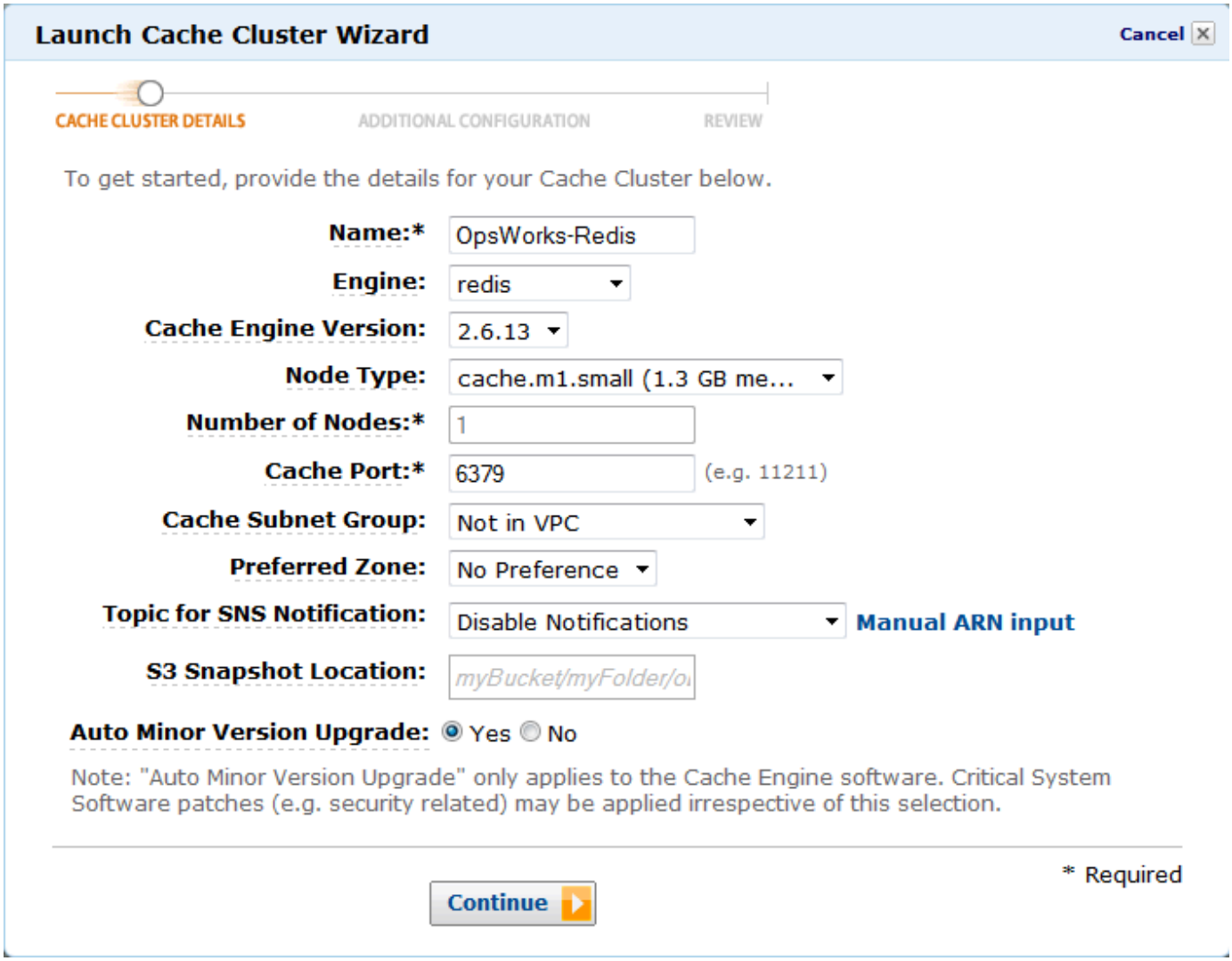
> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You must first create an Amazon ElastiCache Redis cluster by using the ElastiCache console, API, or CLI. The following describes how to use the console to create a cluster.

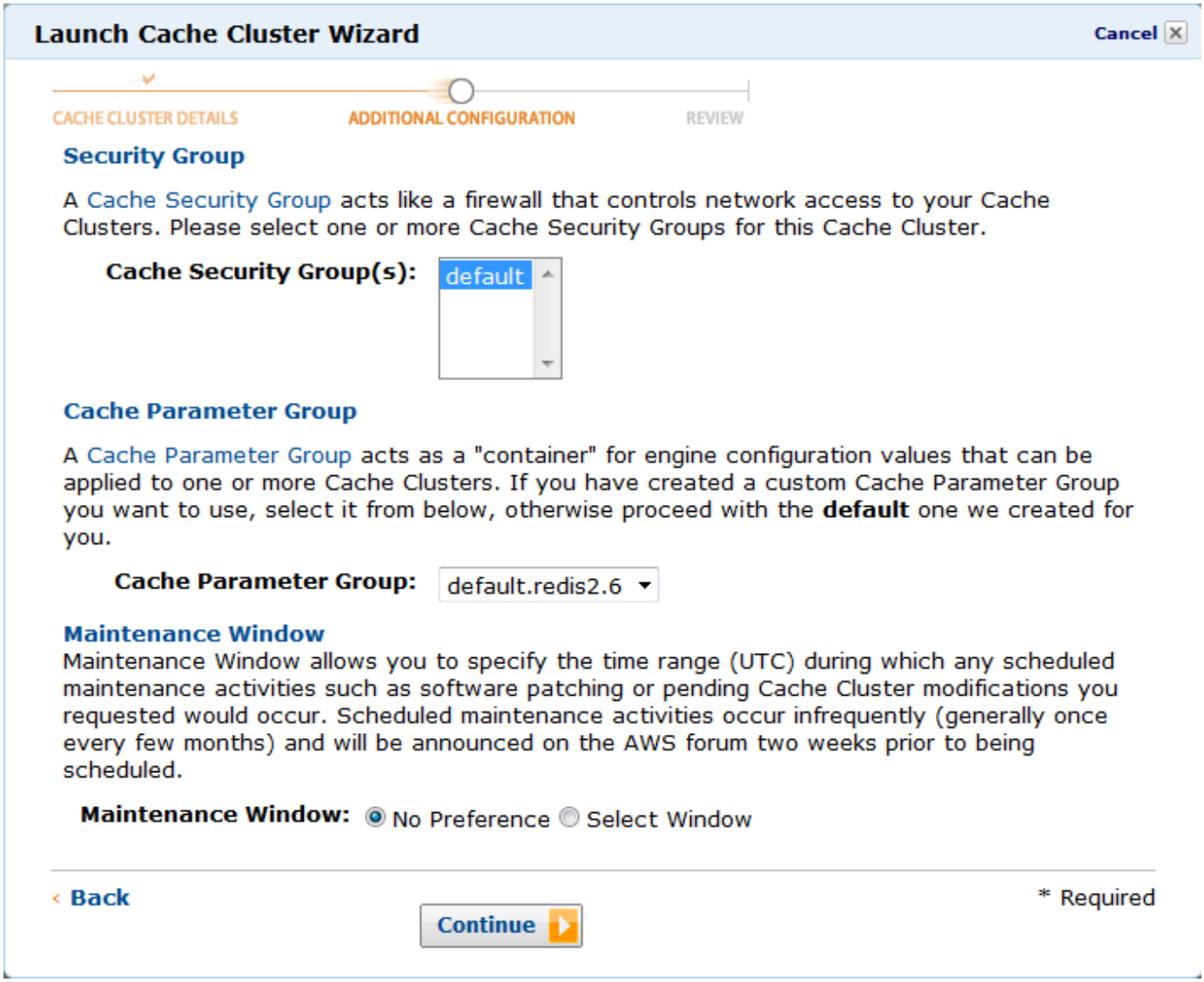
To create an ElastiCache Redis cluster

1. Go to the ElastiCache console and click **Launch Cache Cluster** to start the Cache Cluster wizard.
2. On the Cache Cluster Details page, do the following:
   - Set **Name** to your cache server name.

     This example uses OpsWorks-Redis.
   - Set **Engine** to **redis**.
   - Set **Topic for SNS Notification** to **Disable Notifications**.
   - Accept the defaults for the other settings and click **Continue**.



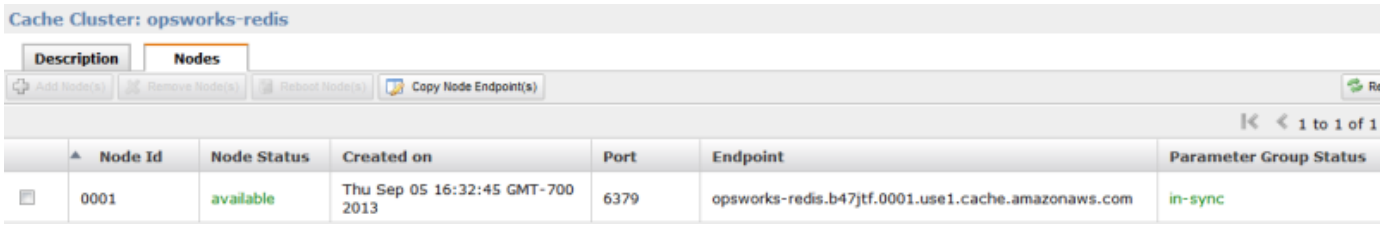
3. On the **Additional Configuration** page, accept the defaults and click **Continue**.



4. Click **Launch Cache Cluster** to create the cluster.

> **Important**
>
> The default cache security group is sufficient for this example, but for production use you should create one that is appropriate for your environment. For more information, see Managing Cache Security Groups.

5. After the cluster has started, click the name to open the details page and click the **Nodes** tab. Record the cluster's **Port** and **Endpoint** values for later use.



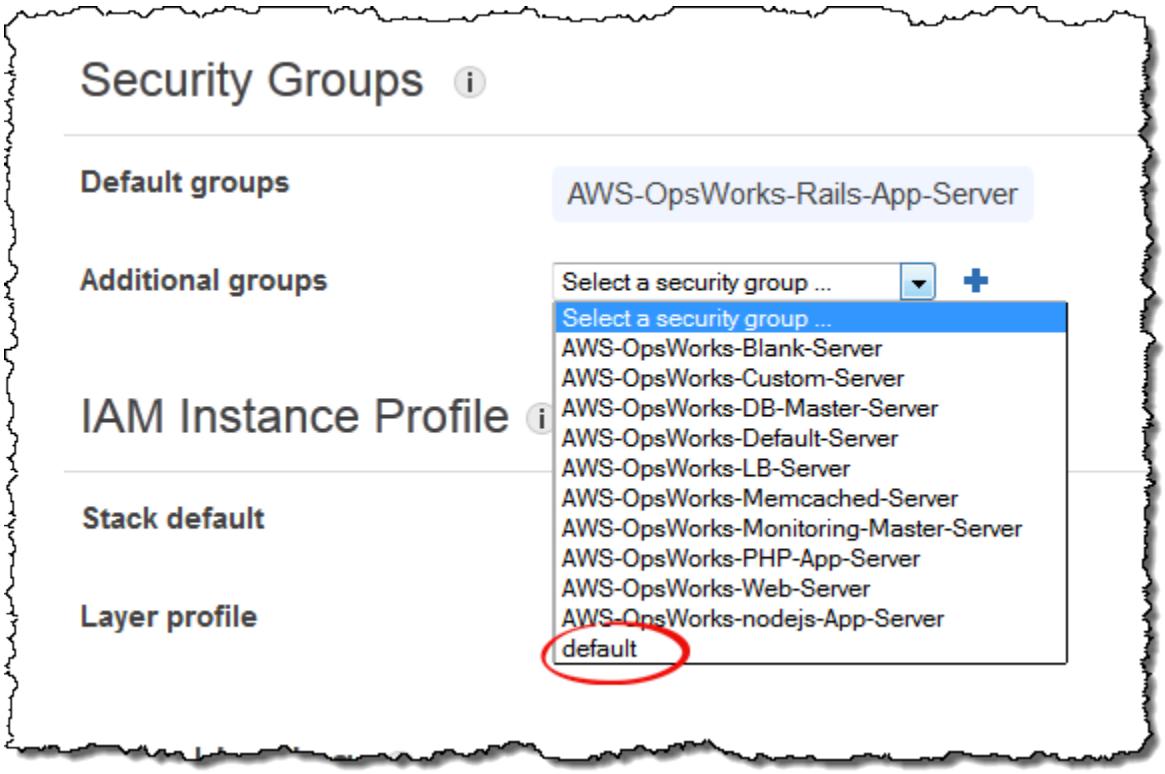
Step 2: Set up a Rails Stack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

In addition to creating a stack that supports a Rails App Server layer, you must also configure the layer's security groups so that the Rails server can communicate properly with Redis server.

To set up a stack

1. Create a new stack—named **`RedisStack`** for this example—and add a Rails App Server layer. You can use the default settings for both. For more information, see Create a New Stack and Creating an OpsWorks Layer .
2. On the **Layers** page, for Rails App Server, click **Security** and then click **Edit**.
3. Go to the **Security Groups** section and add the ElastiCache cluster's security group to **Additional groups**. For this example, select the **default** security group, click **+** to add it to the layer, and click **Save** to save the new configuration.



4. Add an instance to the Rails App Server layer and start it. For more information on how to add and start instances, see Adding an Instance to a Layer.

Step 3: Create and Deploy a Custom Cookbook

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

As it stands, the stack is not quite functional yet; you need to enable your application to access the Redis server. The most flexible approach is to put a YAML file with the access information in the application's `config` subfolder. The application can then get the information from the file. Using this approach, you can change the connection information without rewriting and redeploying the application. For this example, the file should be named `redis.yml` and contain the ElastiCache cluster's host name and port, as follows:

```
host: cache-cluster-hostname
port: cache-cluster-port
```

You could manually copy this file to your servers, but a better approach is to implement a Chef *recipe* to generate the file, and have OpsWorks Stacks run the recipe on every server. Chef recipes are specialized Ruby applications that OpsWorks Stacks uses to perform tasks on instances such as installing packages or creating configuration files. Recipes are packaged in a *cookbook*, which can contain multiple recipes and related files such as templates for configuration files. The cookbook is placed in a repository, such as GitHub, and must have a standard directory structure. If you don't yet have a custom cookbook repository, see Cookbook Repositories for information on how to set one up.

For this example, add a cookbook named `redis-config` to your cookbook repository with the following contents:

```
my_cookbook_repository
  redis-config
    recipes
      generate.rb
    templates
      default
        redis.yml.erb
```

The `recipes` folder contains a recipe named `generate.rb`, which generates the application's configuration file from `redis.yml.erb`, as follows:

```
node[:deploy].each do |app_name, deploy_config|
  # determine root folder of new app deployment
  app_root = "#{deploy_config[:deploy_to]}/current"

  # use template 'redis.yml.erb' to generate 'config/redis.yml'
  template "#{app_root}/config/redis.yml" do
    source "redis.yml.erb"
    cookbook "redis-config"

    # set mode, group and owner of generated file
    mode "0660"
```

```
    group deploy_config[:group]
    owner deploy_config[:user]

    # define variable “@redis” to be used in the ERB template
    variables(
      :redis => deploy_config[:redis] || {}
    )

    # only generate a file if there is Redis configuration
    not_if do
      deploy_config[:redis].blank?
    end
  end
end
```

The recipe depends on data from the OpsWorks Stacks stack configuration and deployment JSON object, which is installed on each instance and contains detailed information about the stack and any deployed apps. The object's `deploy` node has the following structure:

```
{
   ...
  "deploy": {
    "app1": {
      "application" : "short_name",
      ...
    }
    "app2": {
      ...
    }
    ...
  }
}
```

The deploy node contains a set of embedded JSON objects, one for each deployed app, that is named with the app's short name. Each app object contains a set of attributes that define the app's configuration, such as the document root and application type. For a list of the deploy attributes, see deploy Attributes. Recipes can use Chef attribute syntax to represent stack configuration and deployment JSON values. For example,`[:deploy][:app1][:application]` represents the app1 application's short name.

For each app in `[:deploy]`, the recipe executes the associated code block, where `deploy_config` represents the app attribute. The recipe first sets `app_root` to the app's root directory, `[:deploy][:app_name][:deploy_to]/current`. It then uses a Chef template resource to generate a configuration file from `redis.yml.erb` and place it in the `app_root/config`.

Configuration files are typically created from templates, with many if not most of the settings defined by Chef *attributes*. With attributes you can change settings using custom JSON, as described later, instead of rewriting the template file.The `redis.yml.erb` template contains the following:

```
host: <%= @redis[:host] %>
port: <%= @redis[:port] || 6379 %>
```

The <%... %> elements are placeholders that represent an attribute value.

- `<%= @redis[:host] %>` represents the value of `redis[:host]`, which is the cache cluster's host name.
- `<%= @redis[:port] || 6379 %>` represents the value of the `redis[:port]` or, if that attribute is not defined, the default port value, 6379.

The `template` resource works as follows:

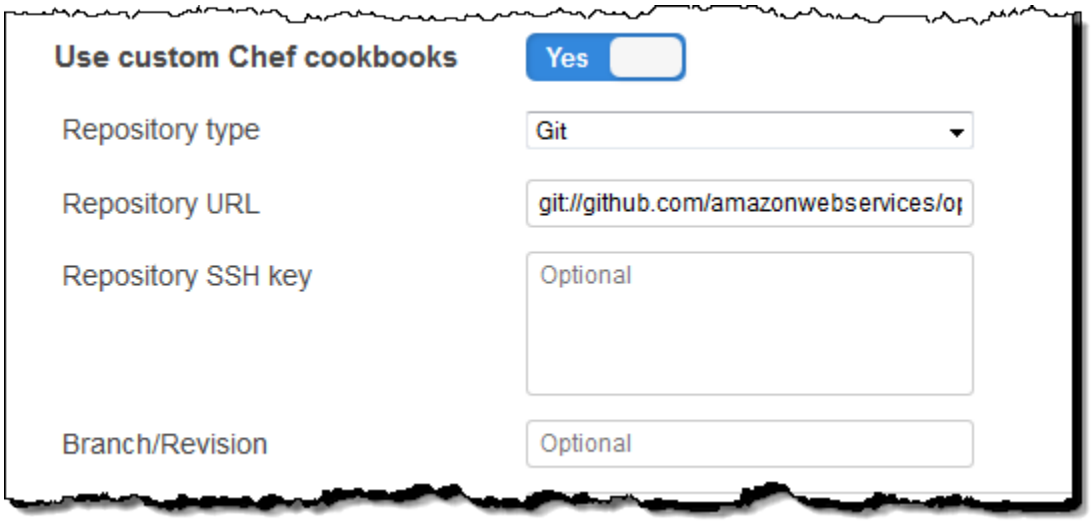
- `source` and `cookbook` specify the template and cookbook names, respectively.
- `mode`, `group`, and `owner` give the configuration file the same access rights as the application.
- The `variables` section sets the `@redis` variable used in the template, to the application's `[:redis]` attribute value.

  The `[:redis]` attribute's values are set by using custom JSON, as described later; it is not one of the standard app attributes.
- The `not_if` directive ensures that the recipe does not generate a configuration file if one already exists.

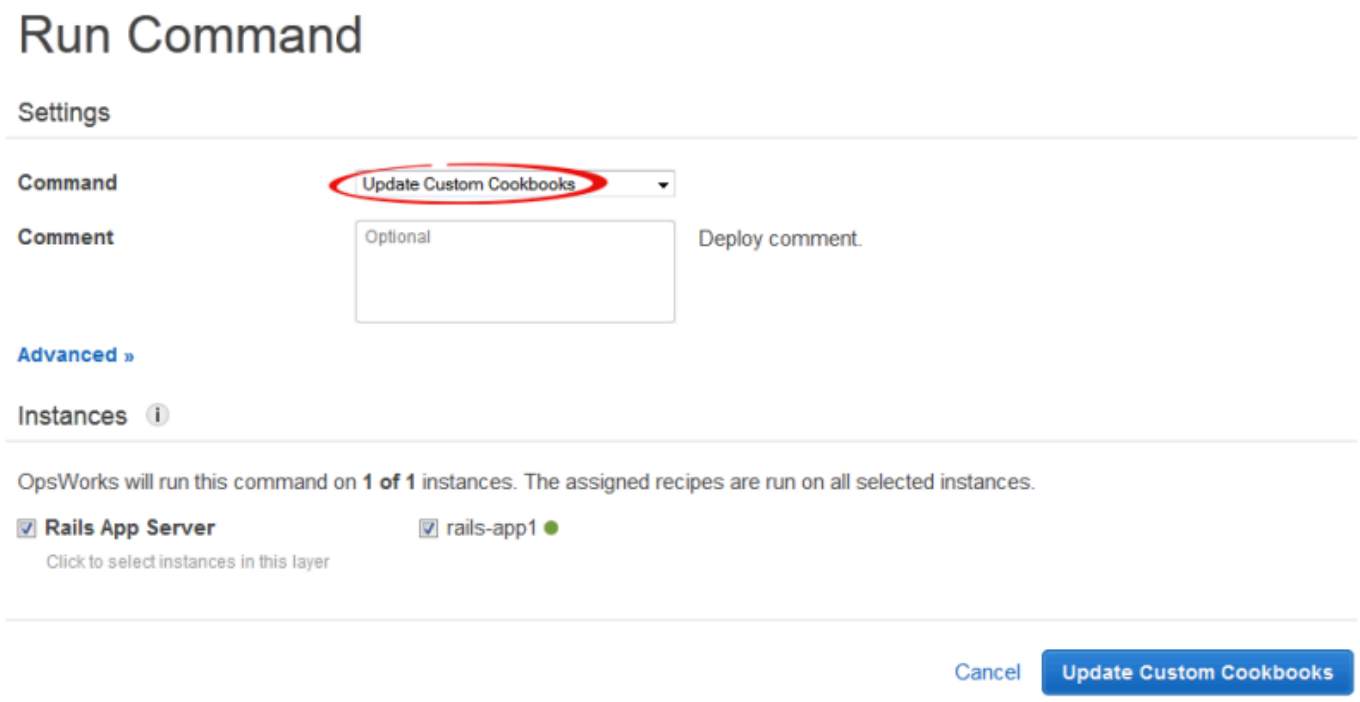
After you author the cookbook, you must deploy it to each instance's cookbook cache. This operation does not run the recipe; it simply installs the new cookbook on the stack's instances. You typically run a recipe by assigning it to a layer's lifecycle event, as described later.

To deploy your custom cookbook

1. On the OpsWorks Stacks **Stack** page, click **Stack Settings** and then **Edit**.
2. In the **Configuration Management** section, set **Use custom Chef cookbooks** to **Yes**, enter the cookbook repository information, and click **Save** to update the stack configuration.



3. On the **Stack** page, click **Run Command**, select the **Update Custom Cookbooks** stack command, and click **Update Custom Cookbooks** to install the new cookbook in the instances' cookbook caches.



If you modify your cookbook, just run **Update Custom Cookbooks** again to install the updated version. For more information on this procedure, see Installing Custom Cookbooks.

Step 4: Assign the Recipe to a LifeCycle Event

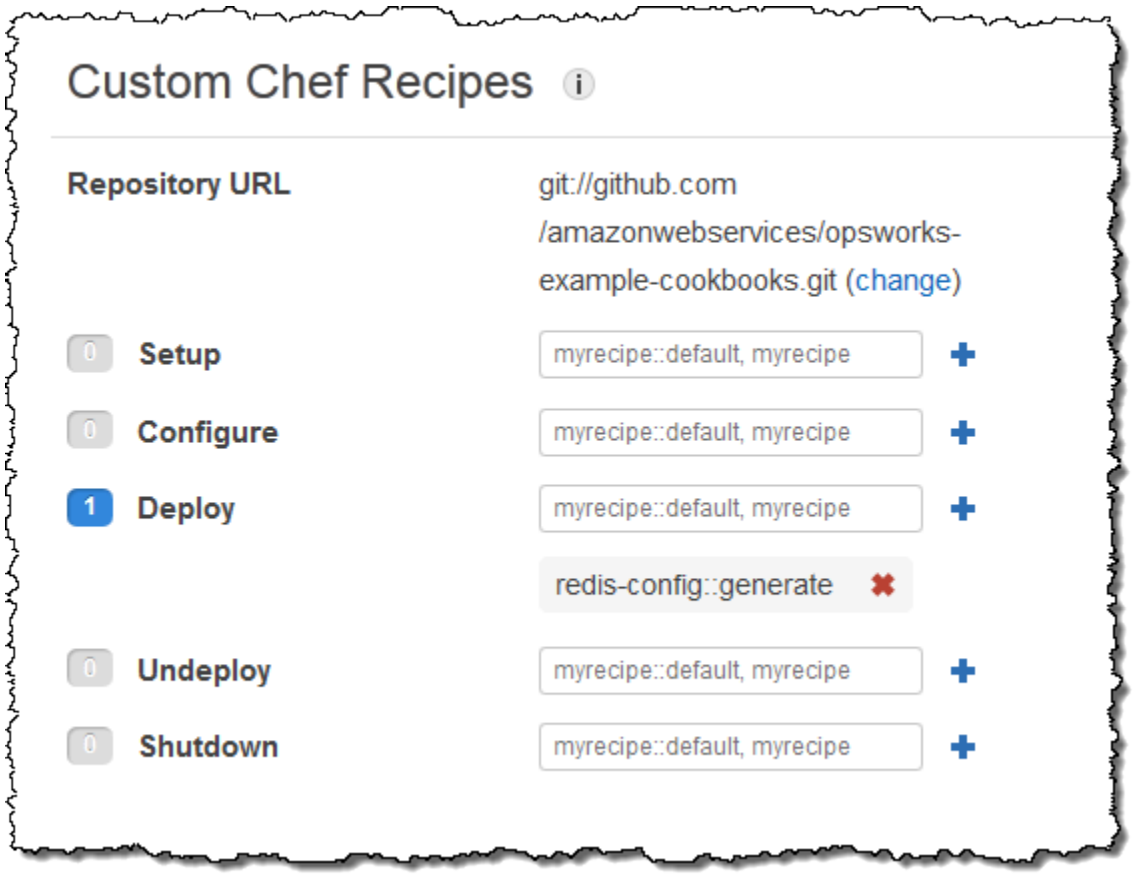
> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can run custom recipes manually, but the best approach is usually to have OpsWorks Stacks run them automatically. Every layer has a set of built-in recipes assigned each of five lifecycle events—Setup, Configure, Deploy, Undeploy, and Shutdown. Each time an event occurs for an instance, OpsWorks Stacks runs the associated recipes for each of the instance's layers, which handle the corresponding tasks. For example, when an instance finishes booting, OpsWorks Stacks triggers a Setup event. This event runs the associated layer's Setup recipes, which typically handle tasks such as installing and configuring packages.

You can have OpsWorks Stacks run a custom recipe on a layer's instances by assigning the recipe to the appropriate lifecycle event. For this example, you should assign the `generate.rb` recipe to the Rails App Server layer's Deploy event. OpsWorks Stacks will then run it on the layer's instances during startup, after the Setup recipes have finished, and every time you deploy an app. For more information, see Automatically Running Recipes.

To assign a recipe to the Rails App Server layer's Deploy event

1. On the OpsWorks Stacks **Layers** page, for Rails App Server, click **Recipes** and then click **Edit**..
2. Under **Custom Chef Recipes**, add the fully qualified recipe name to the deploy event and click **+**. A fully qualified recipe name uses the *`cookbookname`*`::`*`recipename`* format, where *`recipename`* does not include the `.rb` extension. For this example, the fully qualified name is `redis-config::generate`. Then click **Save** to update the layer configuration.



Step 5: Add Access Information to the Stack Configuration JSON

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The `generate.rb` recipe depends on a pair of stack configuration and deployment JSON attributes that represent the Redis server's host name and port. Although these attributes are part of the standard `[:deploy]` namespace, they are not automatically defined by OpsWorks Stacks. Instead, you define the attributes and their values by adding a custom JSON object to the stack. The following example shows the custom JSON for this example.

To add access information to the stack configuration and deployment JSON

1. On the OpsWorks Stacks **Stack** page, click **Stack Settings** and then **Edit**.
2. In the **Configuration Management** section, add access information to the **Custom Chef JSON** box. It should look something like the following example, with these modifications:

   - Replace `elasticache_redis_example` with your app's short name.
   - Replace the `host` and `port` values with the values for the ElastiCache Redis server instance that you created in Step 1: Create an ElastiCache Redis Cluster.

```
{
  "deploy": {
     "elasticache_redis_example": {
       "redis": {
         "host": "mycluster.XXXXXXXXX.amazonaws.com",
         "port": "6379"
       }
     }
  }
}
```



The advantage of this approach is that you can change the port or host value at any time without touching your custom cookbook. OpsWorks Stacks merges custom JSON into the built-in JSON and installs it on the stack's instances for all subsequent lifecycle events. Apps can then access the

attribute values by using Chef node syntax, as described in Step 3: Create and Deploy a Custom Cookbook. The next time you deploy an app, OpsWorks Stacks will install a stack configuration and deployment JSON that contains the new definitions, and `generate.rb` will create a configuration file with the updated host and port values.

> **Note**
>
> `[:deploy]` automatically includes an attribute for every deployed app, so `[:deploy][elasticache_redis_example]` is already in the stack and configuration JSON. However, `[:deploy][elasticache_redis_example]` does not include a `[:redis]` attribute, defining them with custom JSON directs OpsWorks Stacks to add those attributes to `[:deploy][elasticache_redis_example]`. You can also use custom JSON to override existing attributes. For more information, see Overriding Attributes.

Step 6: Deploy and run the App

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This example assumes that you have Ruby on Rails application that uses Redis. To access the configuration file, you can add the `redis` gem to your Gemfile and create a Rails initializer in `config/initializers/redis.rb` as follows:

```
REDIS_CONFIG = YAML::load_file(Rails.root.join('config', 'redis.yml'))
$redis = Redis.new(:host => REDIS_CONFIG['host'], :port => REDIS_CONFIG['port'])
```

Then create an app to represent your application and deploy it to the Rails App Server layer's instances, which updates the application code and runs `generate.rb` to generate the configuration file. When you run the application, it will use the ElastiCache Redis instance as its in-memory key-value store.

Using an Amazon S3 Bucket

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Applications often use an Amazon Simple Storage Service (Amazon S3) bucket to store large items such as images or other media files. Although OpsWorks Stacks does not provide integrated support for Amazon S3, you can easily customize a stack to allow your application to use Amazon S3 storage. This topic walks you through the basic process of providing Amazon S3 access to applications, using a Linux stack with a PHP application server as an example. The basic principles also apply to Windows stacks.

Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

Topics

- Step 1: Create an Amazon S3 Bucket
- Step 2: Create a PHP App Server Stack
- Step 3: Create and Deploy a Custom Cookbook
- Step 4: Assign the Recipes to LifeCycle Events
- Step 5: Add Access Information to the Stack Configuration and Deployment Attributes
- Step 6: Deploy and Run PhotoApp

Step 1: Create an Amazon S3 Bucket

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate

their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You must first create an Amazon S3 bucket. You can do this directly by using the Amazon S3 console, API, or CLI, but a simpler way to create resources is often to use a AWS CloudFormation template. The following template creates an Amazon S3 bucket for this example and sets up instance profile with an IAM role that grants unrestricted access to the bucket. You can then use a layer setting to attach the instance profile to the stack's application server instances, which allows the application to access the bucket, as described later. The usefulness of instance profiles isn't limited to Amazon S3; they are valuable for integrating a variety of AWS services.

```
{
   "AWSTemplateFormatVersion" : "2010-09-09",
   "Resources" : {
      "AppServerRootRole": {
         "Type": "AWS::IAM::Role",
         "Properties": {
            "AssumeRolePolicyDocument": {
               "Statement": [ {
                  "Effect": "Allow",
                  "Principal": {
                     "Service": [ "ec2.amazonaws.com" ]
                  },
                  "Action": [ "sts:AssumeRole" ]
               } ]
            },
            "Path": "/"
         }
      },
      "AppServerRolePolicies": {
         "Type": "AWS::IAM::Policy",
         "Properties": {
            "PolicyName": "AppServerS3Perms",
            "PolicyDocument": {
               "Statement": [ {
                  "Effect": "Allow",
                  "Action": "s3:*",
```

```
                  "Resource": { "Fn::Join" : ["", [ "arn:aws:s3:::", { "Ref" :
 "AppBucket" } , "/*" ]
                  ] }
               } ]
            },
            "Roles": [ { "Ref": "AppServerRootRole" } ]
         }
      },
      "AppServerInstanceProfile": {
         "Type": "AWS::IAM::InstanceProfile",
         "Properties": {
            "Path": "/",
            "Roles": [ { "Ref": "AppServerRootRole" } ]
         }
      },
     "AppBucket" : {
      "Type" : "AWS::S3::Bucket"
      }
   },
   "Outputs" : {
       "BucketName" : {
           "Value" : { "Ref" : "AppBucket" }
       },
       "InstanceProfileName" : {
           "Value" : { "Ref" : "AppServerInstanceProfile" }
       }
   }
}
```

Several things happen when you launch the template:

- The `AWS::S3::Bucket` resource creates an Amazon S3 bucket.
- The `AWS::IAM::InstanceProfile` resource creates an instance profile that will be assigned to the application server instances.
- The `AWS::IAM::Role` resource creates the instance profile's role.
- The `AWS::IAM::Policy` resource sets the role's permissions to allow unrestricted access to Amazon S3 buckets.
- The `Outputs` section displays the bucket and instance profile names in AWS CloudFormation console after you have launched the template.

  You will need these values to set up your stack and app.

For more information on how to create AWS CloudFormation templates, see Learn Template Basics.

To create the Amazon S3 bucket

1. Copy the example template to a text file on your system.

   This example assumes that the file is named `appserver.template`.
2. Open the AWS CloudFormation console and choose **Create Stack**.
3. In the **Stack Name** box, enter the stack name.

   This example assumes that the name is **`AppServer`**.
4. Choose **Upload template file**, choose **Browse**, select the `appserver.template` file that you created in Step 1, and then choose **Next Step**.
5. On the **Specify Parameters** page, select **I acknowledge that this template may create IAM resources**, then choose **Next Step** on each page of the wizard until you reach the end. Choose **Create**.
6. After the **AppServer** stack reaches **CREATE_COMPLETE** status, select it and choose the **Outputs** tab.

   You might need to refresh a few times to update the status.
7. On the **Outputs** tab, record the **BucketName** and **InstanceProfileName** values for later use.

> **Note**
>
> AWS CloudFormation uses the term *stack* to refer to the collection of resources that are created from a template; it is not the same as an OpsWorks Stacks stack.

Step 2: Create a PHP App Server Stack

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

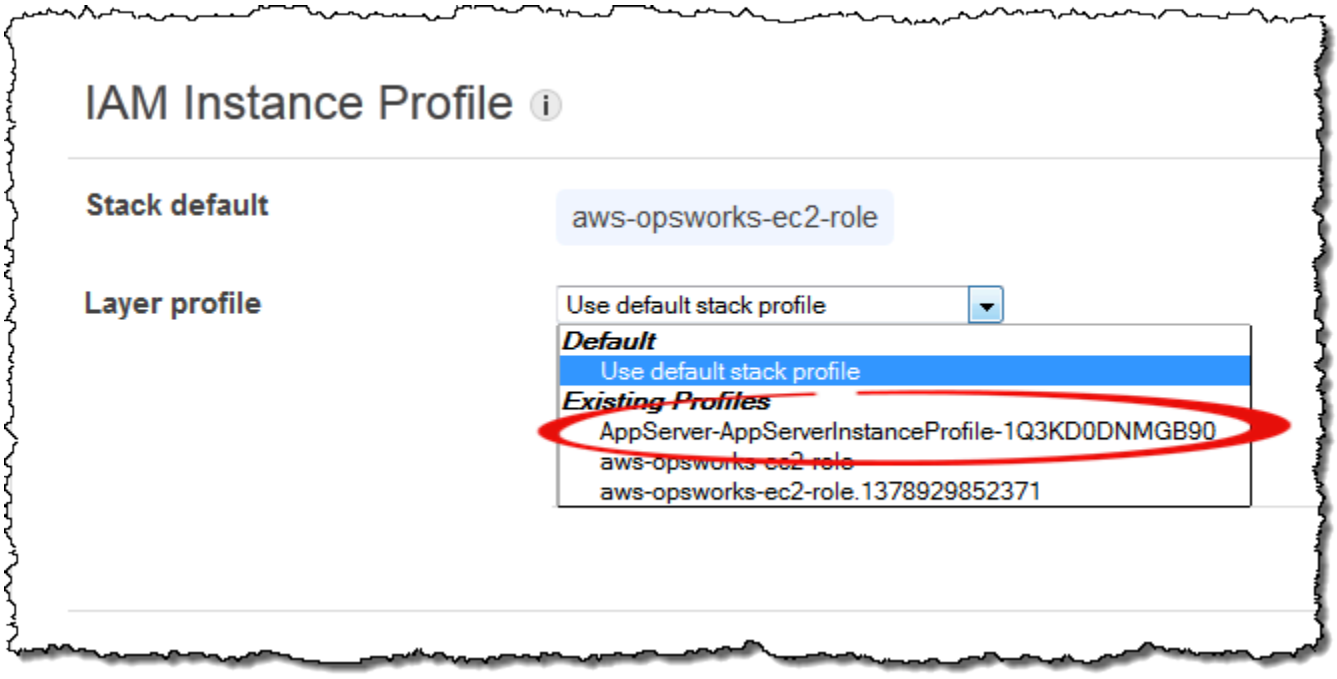
migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The stack consists of two layers, PHP App Server and MySQL, each with one instance. The application stores photos on an Amazon S3 bucket, but uses the MySQL instance as a back-end data store to hold metadata for each photo.

Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

To create the stack

1. Create a new stack—named **`PhotoSite`** for this example—and add a PHP App Server layer. You can use the default settings for both. For more information, see Create a New Stack and Creating an OpsWorks Layer .
2. On the **Layers** page, for PHP App Server, choose **Security** and then choose **Edit**.
3. In the **Layer Profile** section, select the instance profile name that you recorded earlier, after launching the AppServer AWS CloudFormation stack. It will be something like `AppServer-AppServerInstanceProfile-1Q3KD0DNMGB90`. OpsWorks Stacks assigns this profile to all of the layer's Amazon EC2 instances, which grants permission to access your Amazon S3 bucket to applications running on the layer's instances .



4. Add an instance to the PHP App Server layer and start it. For more information on how to add and start instances, see Adding an Instance to a Layer.
5. Add a MySQL layer to the stack, add an instance, and start it. You can use default settings for both the layer and instance. In particular, the MySQL instance doesn't need to access the Amazon S3 bucket, so it can use the standard OpsWorks Stacks instance profile, which is selected by default.

Step 3: Create and Deploy a Custom Cookbook

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The stack is not quite ready yet:

- Your application needs some information to access to the MySQL database server and the Amazon S3 bucket, such as the database host name and the Amazon S3 bucket name.
- You need to set up a database in the MySQL database server and create a table to hold the photos' metadata.

You could handle these tasks manually, but a better approach is to implement Chef *recipe* and have OpsWorks Stacks run the recipe automatically on the appropriate instances. Chef recipes are specialized Ruby applications that OpsWorks Stacks uses to perform tasks on instances such as installing packages or creating configuration files. They are packaged in a *cookbook*, which can contain multiple recipes and related files such as templates for configuration files. The cookbook is placed in a repository such as GitHub, and must have a standard directory structure. If you don't yet have a custom cookbook repository, see Cookbook Repositories for information on how to set one up.

For this example, the cookbook has been implemented for you and is stored in a public GitHub repository. The cookbook contains two recipes, `appsetup.rb` and `dbsetup.rb`, and a template file, `db-connect.php.erb`.

The `appsetup.rb` recipe creates a configuration file that contains the information that the application needs to access the database and the Amazon S3 bucket. It is basically a lightly modified version of the `appsetup.rb` recipe described in Connect the Application to the Database. The primary difference is the variables that are passed to the template, which represent the access information.

The first four attributes define database connection settings, and are automatically defined by OpsWorks Stacks when you create the MySQL instance.

There are two differences between these variables and the ones in the original recipe:

- Like the original recipe, the `table` variable represents the name of the database table that is created by `dbsetup.rb`, and is set to the value of an attribute that is defined in the cookbook's attributes file.

  However, the attribute has a different name: `[:photoapp][:dbtable]`.
- The `s3bucket` variable is specific to this example and is set to the value of an attribute that represents the Amazon S3 bucket name, `[:photobucket]`.

  `[:photobucket]` is defined by using custom JSON, as described later. For more information on attributes, see Attributes

For more information on attributes, see Attributes.

The `dbsetup.rb` recipe sets up a database table to hold each photo's metadata. It basically is a lightly modified version of the `dbsetup.rb` recipe described in Set Up the Database; see that topic for a detailed description.

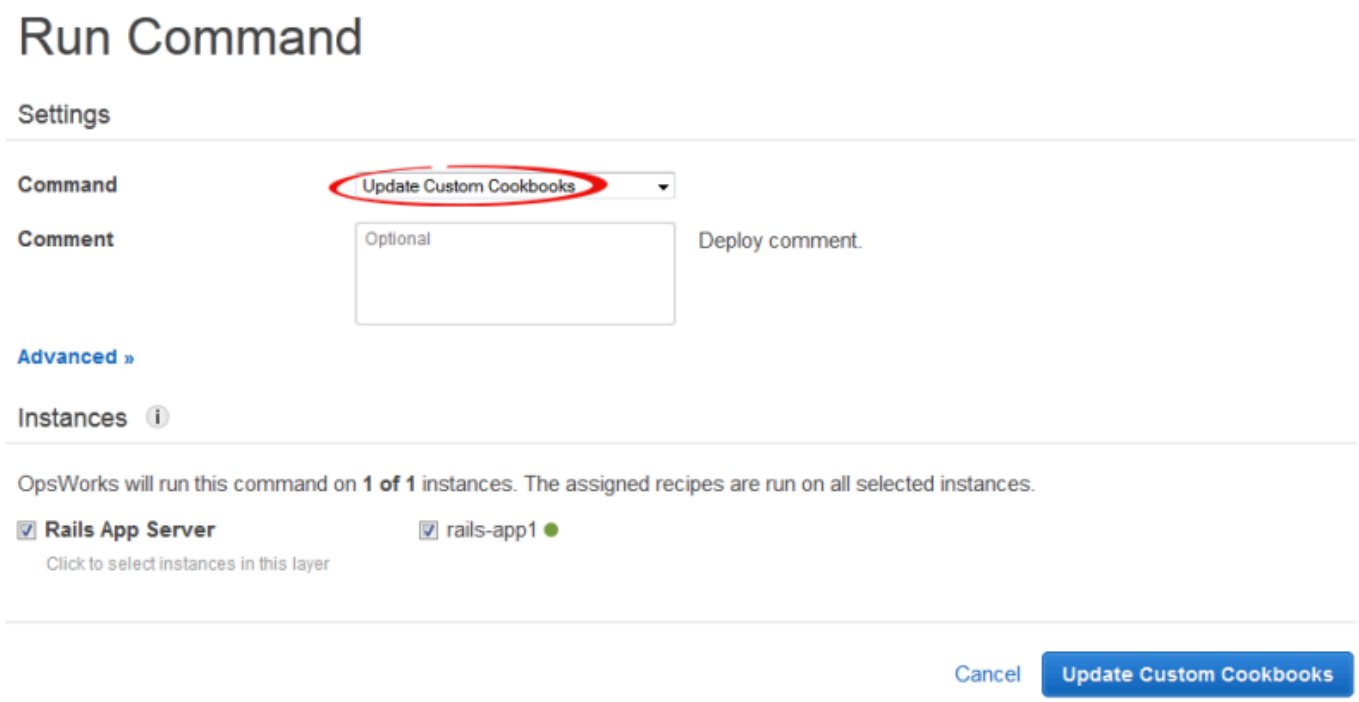
The only difference between this example and the original recipe is the database schema, which has three columns that contain the ID, URL, and caption of each photo that is stored on the Amazon S3 bucket.

The recipes are already implemented, so all you need to do is deploy the photoapp cookbook to each instance's cookbook cache. OpsWorks Stacks then runs the cached recipes when the appropriate lifecycle event occurs, as described later.

To deploy the photoapp cookbook

1. On the OpsWorks Stacks **Stack** page, choose **Stack Settings** and then choose **Edit**.

2. In the **Configuration Management** section:
   - Set **Use custom Chef cookbooks** to **Yes**.
   - Set **Repository type** to Git.
   - Set **Repository URL** to **`git://github.com/amazonwebservices/opsworks-example-cookbooks.git`**.
3. On the **Stack** page, choose **Run Command**, select the **Update Custom Cookbooks** stack command, and then choose **Update Custom Cookbooks** to install the new cookbook in instance cookbook caches.



Step 4: Assign the Recipes to LifeCycle Events

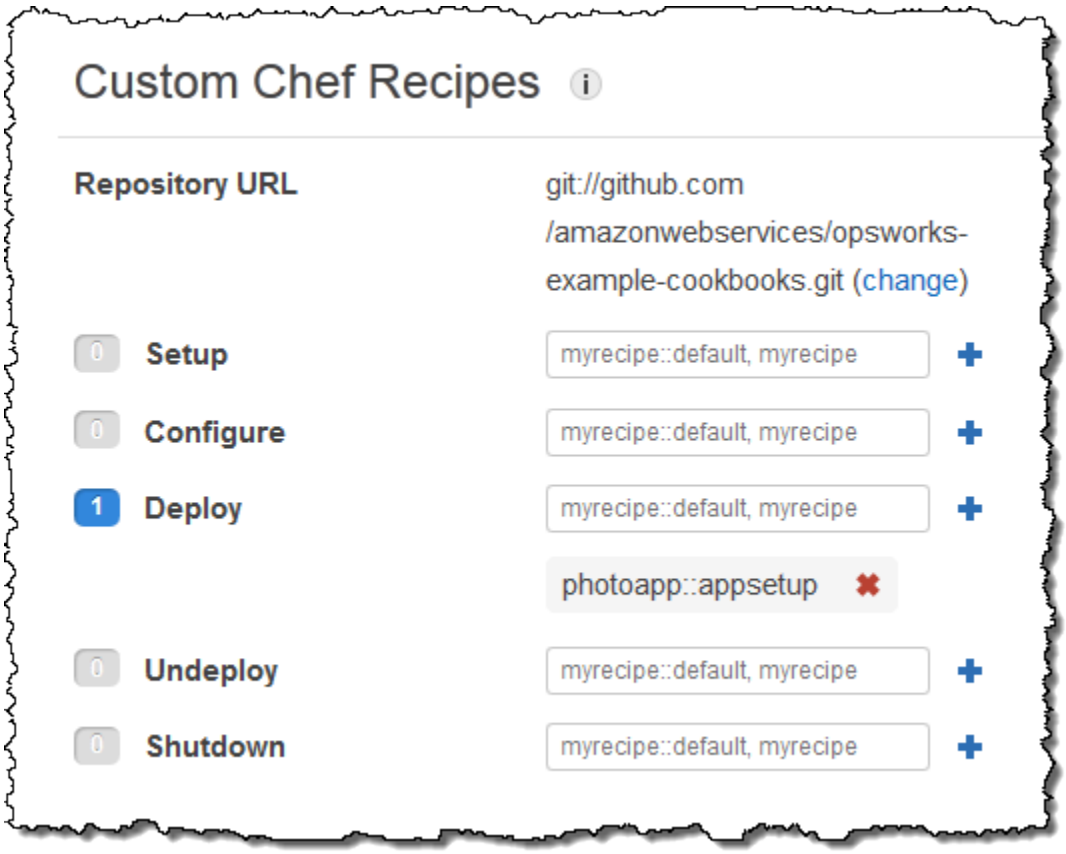
> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

You can run custom recipes manually, but the best approach is usually to have OpsWorks Stacks run them automatically. Every layer has a set of built-in recipes assigned to each of five lifecycle events—Setup, Configure, Deploy, Undeploy, and Shutdown—. Each time an event occurs on an instance, OpsWorks Stacks runs the associated recipes for each of the instance's layers, which handle the required tasks. For example, when an instance finishes booting, OpsWorks Stacks triggers a Setup event to run the Setup recipes, which typically handle tasks such as installing and configuring packages .

You can have OpsWorks Stacks run custom recipes on a layer's instances by assigning each recipe to the appropriate lifecycle event. OpsWorks Stacks will run any custom recipes after the layer's built-in recipes have finished. For this example, assign `appsetup.rb` to the PHP App Server layer's Deploy event and `dbsetup.rb` to the MySQL layer's Deploy event. OpsWorks Stacks will then run the recipes on the associated layer's instances during startup, after the built-in Setup recipes have finished, and every time you deploy an app, after the built Deploy recipes have finished. For more information, see Automatically Running Recipes.

To assign custom recipes to the layer's Deploy event

1. On the OpsWorks Stacks **Layers** page, for the PHP App Server choose **Recipes** and then choose **Edit**.
2. Under **Custom Chef Recipes**, add the recipe name to the deploy event and choose **+**. The name must be in the Chef *`cookbookname`*`::`*`recipename`* format, where *`recipename`* does not include the `.rb` extension. For this example, you enter `photoapp::appsetup`. Then choose **Save** to update the layer configuration.



3. On the **Layers** page, choose **edit** in the MySQL layer's **Actions** column.
4. Add `photoapp::dbsetup` to the layer's Deploy event and save the new configuration.

Step 5: Add Access Information to the Stack Configuration and Deployment Attributes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The `appsetup.rb` recipe depends on data from the OpsWorks Stacks stack configuration and deployment attributes, which are installed on each instance and contain detailed information

about the stack and any deployed apps. The object's `deploy` attributes have the following structure, which is displayed for convenience as JSON:

```
{
   ...
  "deploy": {
    "app1": {
      "application" : "short_name",
      ...
    }
    "app2": {
      ...
    }
    ...
  }
}
```

The deploy node contains an attribute for each deployed app that is named with the app's short name. Each app attribute contains a set of attributes that define the app's configuration, such as the document root and app type. For a list of the `deploy` attributes, see deploy Attributes. You can represent stack configuration and deployment attribute values in your recipes by using Chef attribute syntax. For example,`[:deploy][:app1][:application]` represents the app1 app's short name.

The custom recipes depend on several stack configuration and deployment attributes that represent database and Amazon S3 access information:

- The database connection attributes, such as `[:deploy][:database][:host]`, are defined by OpsWorks Stacks when it creates the MySQL layer.
- The table name attribute, `[:photoapp][:dbtable]`, is defined in the custom cookbook's attributes file, and is set to `foto`.
- You must define the bucket name attribute, `[:photobucket]`, by using custom JSON to add the attribute to the stack configuration and deployment attributes.

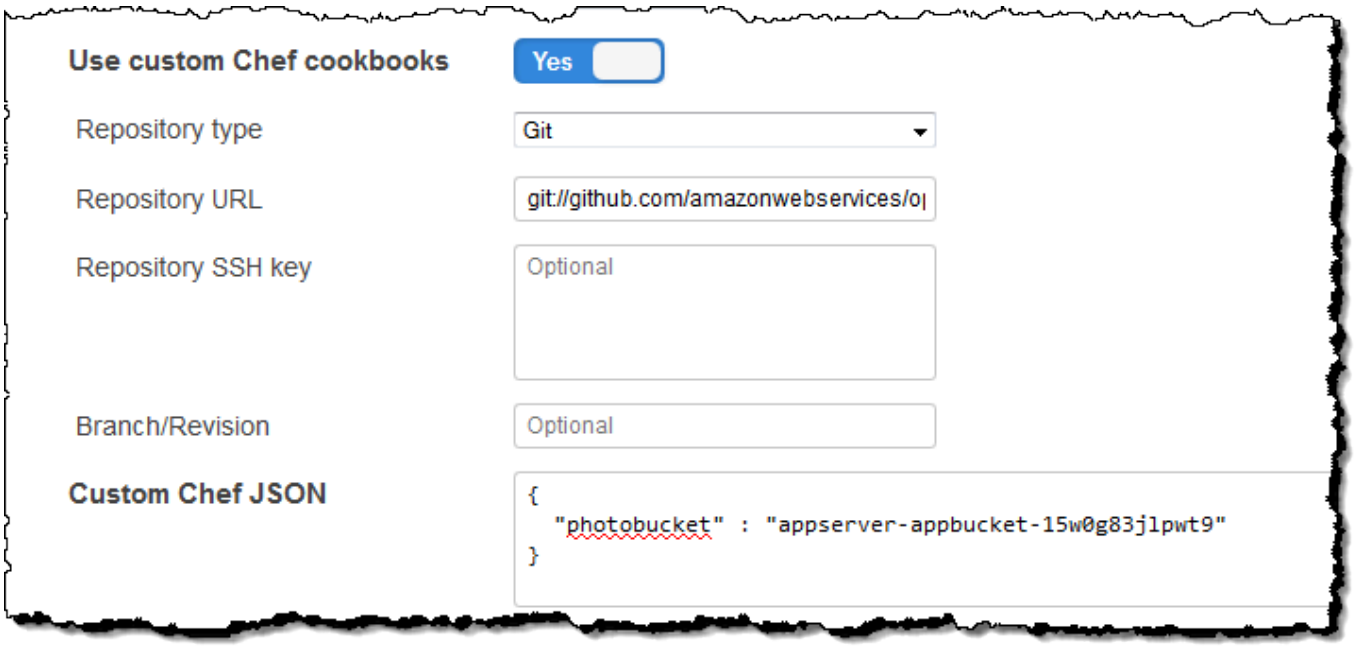
To define the Amazon S3 bucket name attribute

1. On the OpsWorks Stacks **Stack** page, choose **Stack Settings** and then **Edit**.

2. In the **Configuration Management** section, add access information to the **Custom Chef JSON** box. It should look something like the following:

```
{
  "photobucket" : "yourbucketname"
}
```

Replace *yourbucketname* with the bucket name that you recorded in Step 1: Create an Amazon S3 Bucket.



OpsWorks Stacks merges the custom JSON into the stack configuration and deployment attributes before it installs them on the stack's instances; `appsetup.rb` can then obtain the bucket name from the `[:photobucket]` attribute. If you want to change the bucket, you don't need to touch the recipe; you can just override the attribute to provide a new bucket name.

Step 6: Deploy and Run PhotoApp

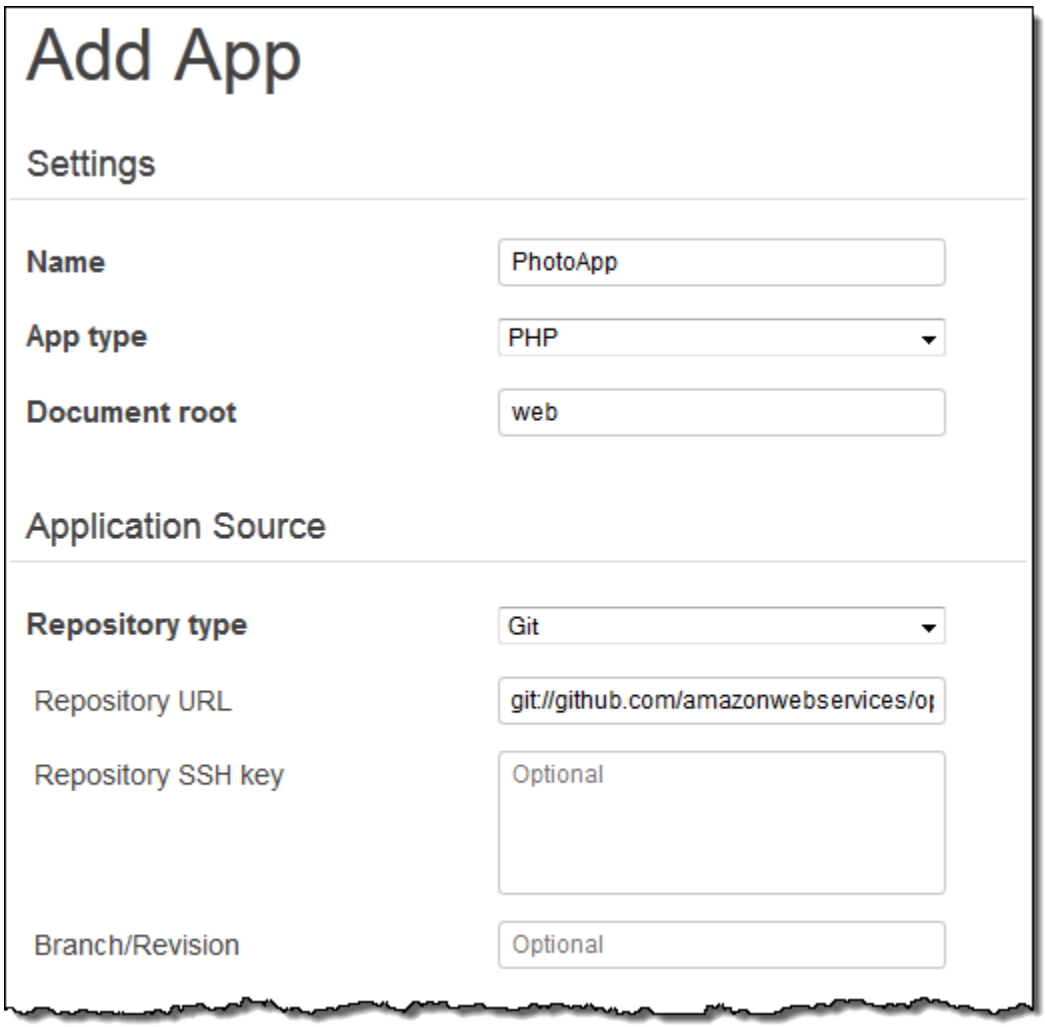
> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

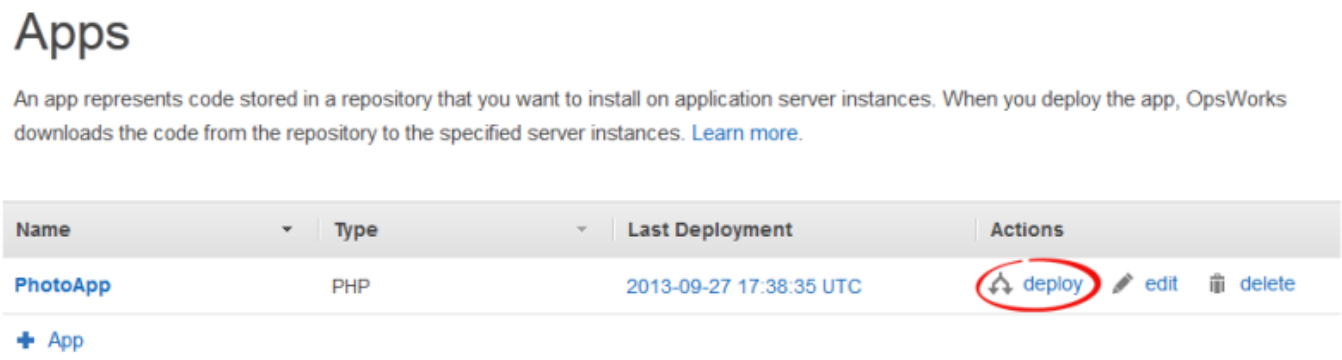
For this example, the application has also been implemented for you and is stored in a public GitHub repository. You just need to add the app to the stack, deploy it to the application servers, and run it.

To add the app to the stack and deploy it to the application servers

1. Open the **Apps** page and choose **Add an app**.
2. On the **Add App** page, do the following:
   - Set **Name** to **PhotoApp**.
   - Set **App type** to **PHP**.
   - Set **Document root** to **web**.
   - Set **Repository type** to **Git**.
   - Set **Repository URL** to **`git://github.com/awslabs/opsworks-demo-php-photo-share-app.git`**.
   - Choose **Add App** to accept the defaults for the other settings.

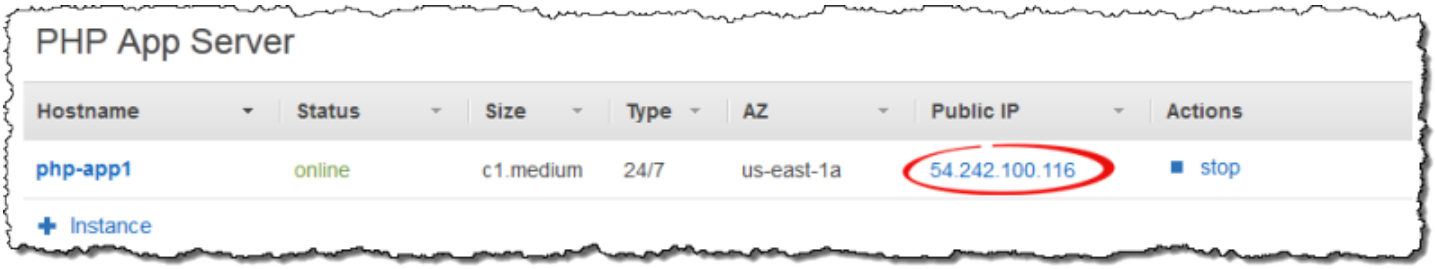


3. On the **Apps** page, choose **deploy** in the PhotoApp app's **Actions** column.



4. Accept the defaults and choose **Deploy** to deploy the app to the server.

To run PhotoApp, go to the **Instances** page and choose the PHP App Server instance's public IP address.



You should see the following user interface. Choose **Add a Photo** to store a photo on the Amazon S3 bucket and the metadata in the back-end data store.



Using AWS CodePipeline with OpsWorks Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

AWS CodePipeline lets you create continuous delivery pipelines that track code changes from sources such as CodeCommit, Amazon Simple Storage Service (Amazon S3), or GitHub. You can use CodePipeline to automate the release of your Chef cookbooks and application code to OpsWorks Stacks, on Chef 11.10, Chef 12, and Chef 12.2 stacks. Examples in this section describe how to create and use a simple pipeline from CodePipeline as a deployment tool for code that you run on OpsWorks Stacks layers.

> **Note**
>
> CodePipeline and OpsWorks Stacks integration is not supported for deploying to Chef 11.4 and older stacks.

Topics

- AWS CodePipeline with OpsWorks Stacks - Chef 12 Stacks
- AWS CodePipeline with OpsWorks Stacks - Chef 11 Stacks

AWS CodePipeline with OpsWorks Stacks - Chef 12 Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

AWS CodePipeline lets you create continuous delivery pipelines that track code changes from sources such as CodeCommit, Amazon Simple Storage Service (Amazon S3), or GitHub. The example in this topic describes how to create and use a simple pipeline from CodePipeline as a deployment tool for code that you run on OpsWorks Stacks layers. In this example, you create a pipeline for a simple Node.js app, and then instruct OpsWorks Stacks to run the app on all of the instances in a layer in a Chef 12 stack (in this case, a single instance).

> **Note**
>
> This topic describes how to use a pipeline to run and update an app on a Chef 12 stack. For information about how to use a pipeline to run and update an app on a Chef 11.10 stack, see AWS CodePipeline with OpsWorks Stacks - Chef 11 Stacks. Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

Topics

- Prerequisites
- Other Supported Scenarios
- Step 1: Create a stack, layer, and an instance in OpsWorks Stacks
- Step 2: Configure your stack and layer to use custom cookbooks
- Step 3: Upload app code to an Amazon S3 bucket
- Step 4: Add your app to OpsWorks Stacks
- Step 5: Create a pipeline in CodePipeline
- Step 6: Verifying the app deployment in OpsWorks Stacks
- Step 7 (Optional): Update the app code to see CodePipeline redeploy your app automatically
- Step 8 (Optional): Clean up resources

Prerequisites

Before you start this walkthrough, be sure that you have administrator permissions to do all of the following tasks. You can be a member of a group that has the **AdministratorAccess** policy applied, or you can be a member of a group that has the permissions and policies shown in the following table. As a security best practice, you should belong to a group that has permissions to do the following tasks, instead of assigning required permissions to individual users.

For more information about creating a security group in IAM and assigning permissions to the group, see Creating IAM user groups. For more information about managing OpsWorks Stacks permissions, see Best Practices: Managing Permissions.

Permissions	Recommended Policy to Attach to Group
Create and edit stacks, layers, and instances in OpsWorks Stacks.	**AWSOpsWorks_FullAccess**
Create, edit, and run templates in AWS CloudFormation.	**AmazonCloudFormationFullAccess**
Create, edit, and access Amazon S3 buckets.	**AmazonS3FullAccess**

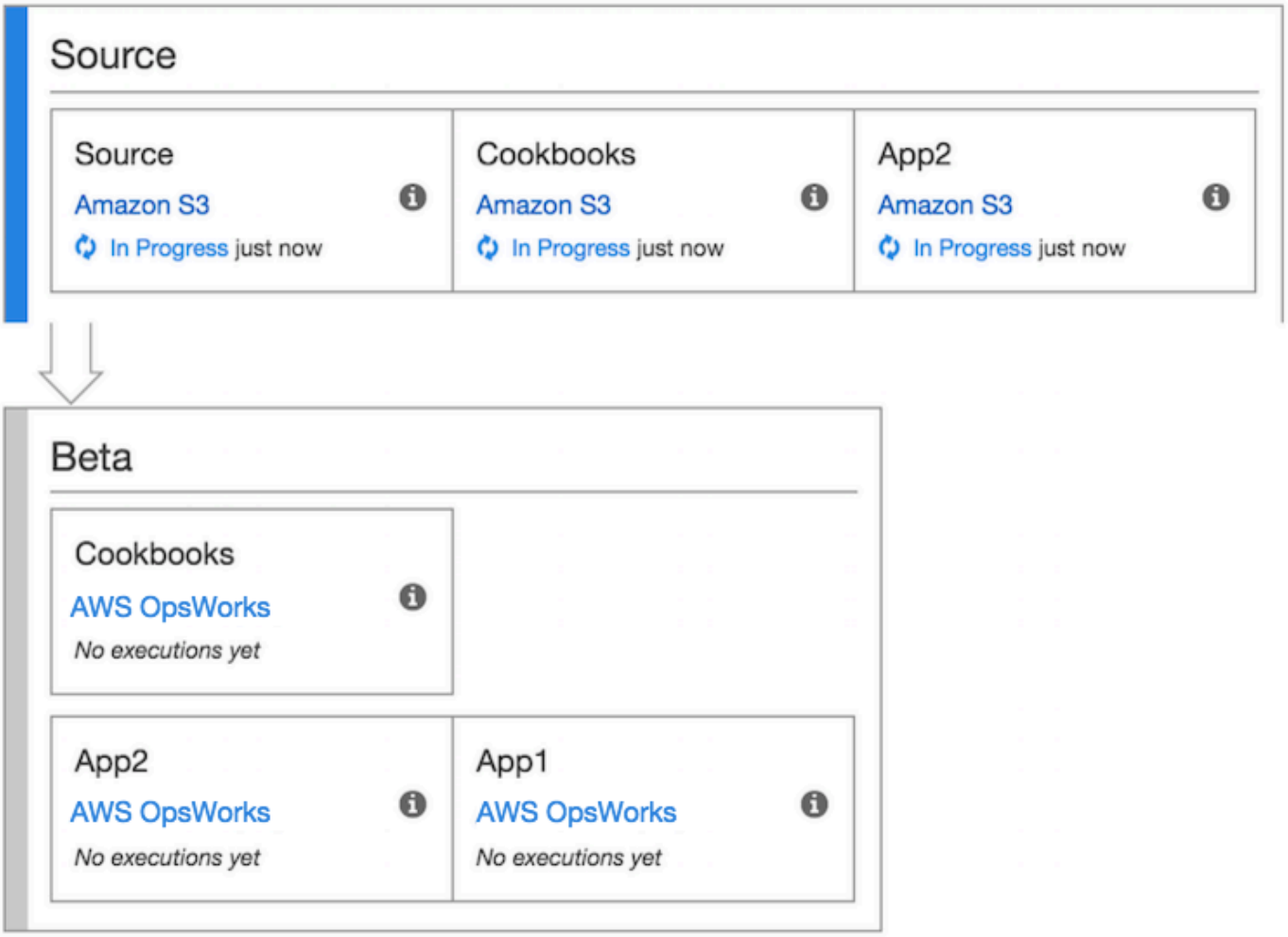
Permissions	Recommended Policy to Attach to Group
Create, edit, and run pipelines in CodePipeline, especially pipelines that use OpsWorks Stacks as the provider.	**AWSCodePipeline_FullAccess**

You must also have an Amazon EC2 key pair. You will be prompted to provide the name of this key pair when you run the AWS CloudFormation template that creates the sample stack, layer, and instance in this walkthrough. For more information about obtaining a key pair in the Amazon EC2 console, see Create a Key Pair in the Amazon EC2 documentation. The key pair must be in the US East (N. Virginia) Region. You can use an existing key pair if you already have one in that region.

Other Supported Scenarios

This walkthrough creates a simple pipeline that includes one **Source** and one **Deploy** stage. However, you can create more complex pipelines that use OpsWorks Stacks as a provider. The following are examples of supported pipelines and scenarios:

- You can edit a pipeline to add a Chef cookbook to the **Source** stage and an associated target for updated cookbooks to the **Deploy** stage. In this case, you add a **Deploy** action that triggers the updating of your cookbooks when you make changes to the source. The updated cookbook is deployed before your app.
- You can create a complex pipeline, with custom cookbooks and multiple apps, and deploy to an OpsWorks Stacks stack. The pipeline tracks changes to both the application and cookbook sources, and redeploys when you have made changes. The following shows an example of a similar, complex pipeline:



For more information about working with CodePipeline, see the CodePipeline User Guide.

Step 1: Create a stack, layer, and an instance in OpsWorks Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

To use OpsWorks Stacks as a deployment provider for a pipeline, you must first have a stack, a layer, and at least one instance in the layer. Although you can create a stack in OpsWorks Stacks by following instructions in Getting Started with Linux Stacks or Getting Started with Windows Stacks, to save you time, this example uses an AWS CloudFormation template to create a Linux-based Chef 12 stack, layer, and instance. The instance created by this template runs Amazon Linux 2016.03, and has an instance type of `c3.large`. Although the template does not configure your stack to use custom cookbooks, you'll do this later in the walkthrough.

> **Important**
>
> The AWS CloudFormation template must be stored and run in the same region as the Amazon S3 bucket to which you later upload your app and the same region in which you later create your pipeline in CodePipeline. At this time, CodePipeline supports the OpsWorks Stacks provider in the US East (N. Virginia) Region (us-east-1) only. All resources in this walkthrough should be created in the US East (N. Virginia) Region.
> If stack creation fails, you might be approaching the maximum allowed number of IAM roles for your account. The stack creation can also fail if your account cannot launch instances with a `c3.large` instance type. For example, if you are using the AWS Free Tier, you might receive an error such as `Root device type: must be included in EBS`. If your account has limitations on the instance types that you are allowed to create, such as limitations imposed by the AWS Free Tier, try changing the value of the `InstanceType` parameter in the template's instance block to an instance type that your account can use.

To create a stack, layer, and instance using AWS CloudFormation

1. Copy the following AWS CloudFormation template into a new plain-text document. Save the file to a convenient location on your local computer, and name it **NewOpsWorksStack.template**, or another name that is convenient for you.

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Mappings": {
    "Region2Principal": {
      "us-east-1": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "us-west-2": {
```

```
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "us-west-1": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "eu-west-1": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "ap-southeast-1": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "ap-northeast-1": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "ap-northeast-2": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "ap-southeast-2": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "sa-east-1": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "cn-north-1": {
        "EC2Principal": "ec2.amazonaws.com.cn",
        "OpsWorksPrincipal": "opsworks.amazonaws.com.cn"
      },
      "eu-central-1": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      }
    }
  },
  "Parameters": {
    "EC2KeyPairName": {
   "Type": "String",
```

```
  "Description": "The name of an existing EC2 key pair that lets you use SSH to
connect to the OpsWorks instance."
 }
 },
 "Resources": {
"CPOpsDeploySecGroup": {
  "Type": "AWS::EC2::SecurityGroup",
  "Properties": {
    "GroupDescription" : "Lets you manage OpsWorks instances to which you deploy
apps with CodePipeline"
  }
},
"CPOpsDeploySecGroupIngressHTTP": {
  "Type": "AWS::EC2::SecurityGroupIngress",
  "Properties" : {
    "IpProtocol" : "tcp",
       "FromPort" : "80",
       "ToPort" : "80",
       "CidrIp" : "0.0.0.0/0",
 "GroupId": {
   "Fn::GetAtt": [
     "CPOpsDeploySecGroup", "GroupId"
   ]
 }
     }
},
"CPOpsDeploySecGroupIngressSSH": {
  "Type": "AWS::EC2::SecurityGroupIngress",
  "Properties" : {
    "IpProtocol" : "tcp",
       "FromPort" : "22",
       "ToPort" : "22",
       "CidrIp" : "0.0.0.0/0",
 "GroupId": {
   "Fn::GetAtt": [
     "CPOpsDeploySecGroup", "GroupId"
   ]
 }
  }
},
"MyStack": {
     "Type": "AWS::OpsWorks::Stack",
     "Properties": {
       "Name": {
```

```
          "Ref": "AWS::StackName"
        },
        "ServiceRoleArn": {
          "Fn::GetAtt": [
            "OpsWorksServiceRole",
            "Arn"
          ]
        },
  "ConfigurationManager" : { "Name": "Chef","Version": "12" },
  "DefaultOs": "Amazon Linux 2016.03",
        "DefaultInstanceProfileArn": {
          "Fn::GetAtt": [
            "OpsWorksInstanceProfile",
            "Arn"
          ]
        },
  "UseCustomCookbooks": "false"
      }
    },
    "MyLayer": {
      "Type": "AWS::OpsWorks::Layer",
      "Properties": {
        "StackId": {
          "Ref": "MyStack"
        },
        "Name": "Node.js App Server",
  "Type": "custom",
        "Shortname": "app1",
  "EnableAutoHealing": "true",
        "AutoAssignElasticIps": "false",
        "AutoAssignPublicIps": "true",
  "CustomSecurityGroupIds": [
    {
      "Fn::GetAtt": [
              "CPOpsDeploySecGroup", "GroupId"
      ]
    }
   ]
      },
      "DependsOn": [
        "MyStack",
        "CPOpsDeploySecGroup"
      ]
    },
```

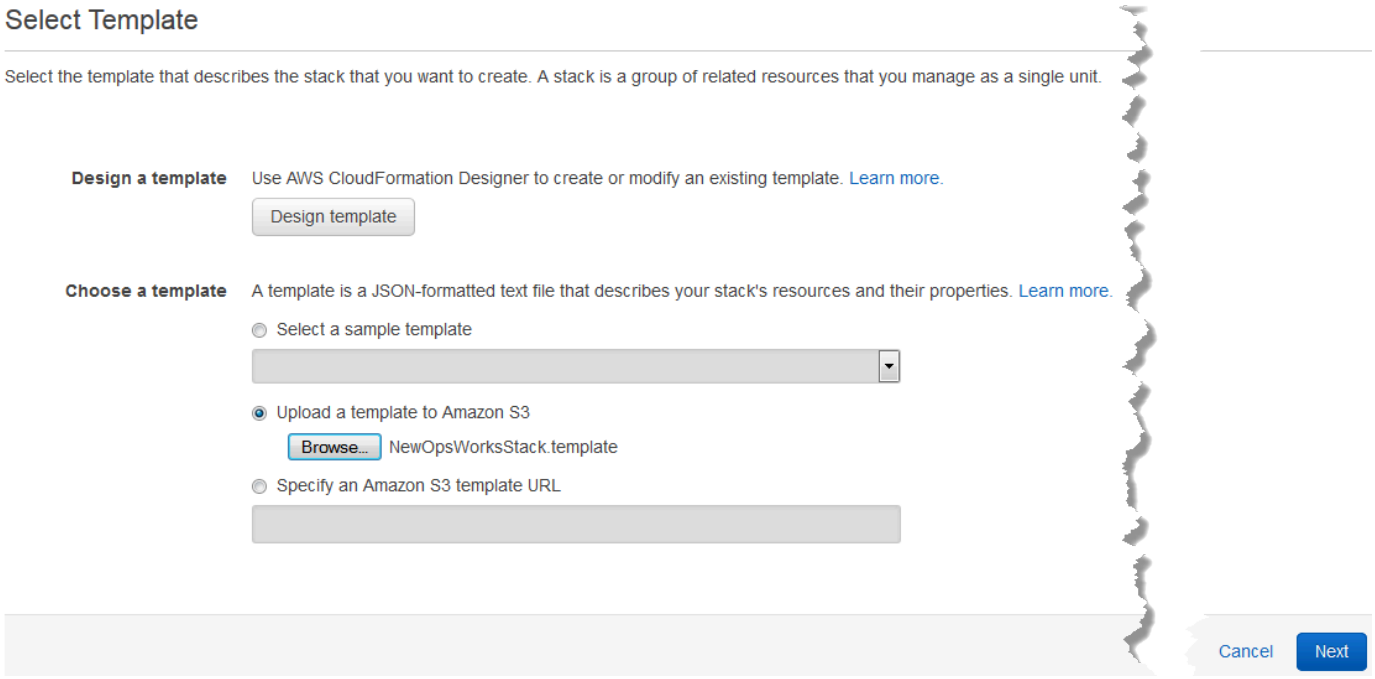
```
    "OpsWorksServiceRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  {
                    "Fn::FindInMap": [
                      "Region2Principal",
                      {
                        "Ref": "AWS::Region"
                      },
                      "OpsWorksPrincipal"
                    ]
                  }
                ]
              },
              "Action": [
                "sts:AssumeRole"
              ]
            }
          ]
        },
        "Path": "/",
        "Policies": [
          {
            "PolicyName": "opsworks-service",
            "PolicyDocument": {
              "Statement": [
                {
                  "Effect": "Allow",
                  "Action": [
                    "ec2:*",
                    "iam:PassRole",
                    "cloudwatch:GetMetricStatistics",
                    "elasticloadbalancing:*"
                  ],
                  "Resource": "*"
                }
              ]
            }
```

```
          }
        ]
      }
    },
    "OpsWorksInstanceProfile": {
      "Type": "AWS::IAM::InstanceProfile",
      "Properties": {
        "Path": "/",
        "Roles": [
          {
            "Ref": "OpsWorksInstanceRole"
          }
        ]
      }
    },
    "OpsWorksInstanceRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  {
                    "Fn::FindInMap": [
                      "Region2Principal",
                      {
                        "Ref": "AWS::Region"
                      },
                      "EC2Principal"
                    ]
                  }
                ]
              },
              "Action": [
                "sts:AssumeRole"
              ]
            }
          ]
        },
        "Path": "/",
  "Policies": [
          {
```

```
            "PolicyName": "s3-get",
            "PolicyDocument": {
              "Version": "2012-10-17",
              "Statement": [
                {
                  "Effect": "Allow",
                  "Action": [
                    "s3:GetObject"
                  ],
                  "Resource": "*"
                }
              ]
            }
          }
        ]
      }
    },
    "myinstance": {
      "Type": "AWS::OpsWorks::Instance",
      "Properties": {
        "LayerIds": [
          {
            "Ref": "MyLayer"
          }
        ],
        "StackId": {
          "Ref": "MyStack"
        },
        "InstanceType": "c3.large",
        "SshKeyName": {
    "Ref": "EC2KeyPairName"
  }
      }
    }
  },
  "Outputs": {
    "StackId": {
      "Description": "Stack ID for the newly created AWS OpsWorks stack",
      "Value": {
        "Ref": "MyStack"
      }
    }
  }
```

```
}
```

2. Sign in to the AWS Management Console and open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
3. On the AWS CloudFormation home page, choose **Create stack**.
4. On the **Select Template** page, in the **Choose a template** area, choose **Upload a template to Amazon S3**, and then choose **Browse**.
5. Browse to the AWS CloudFormation template that you saved in step 1, and then choose **Open**. On the **Select Template** page, choose **Next**.

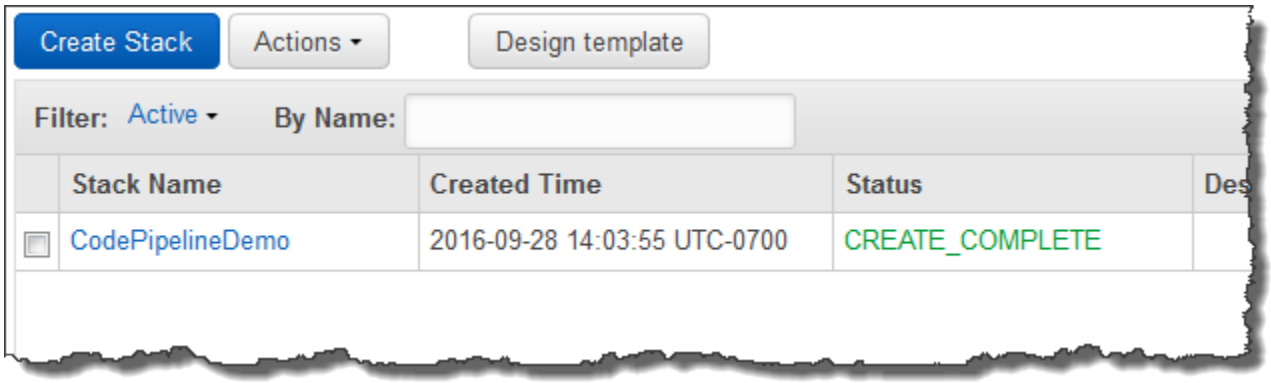


6. On the **Specify Details** page, name the stack **CodePipelineDemo**, or any stack name that is unique to your account. If you choose a different name for your stack, change the stack name throughout this walkthrough.
7. In the **Parameters** area, provide the name of an EC2 key pair that you want to use to access your OpsWorks Stacks instance after it has been created. Choose **Next**.
8. On the **Options** page, choose **Next**. (Settings on this page are not required for this walkthrough.)
9. The AWS CloudFormation template that you use in this walkthrough creates IAM roles, an instance profile, and an instance.

> **Important**
>
> Before you choose **Create**, choose **Cost** to estimate charges you might incur from AWS for creating resources with this template.

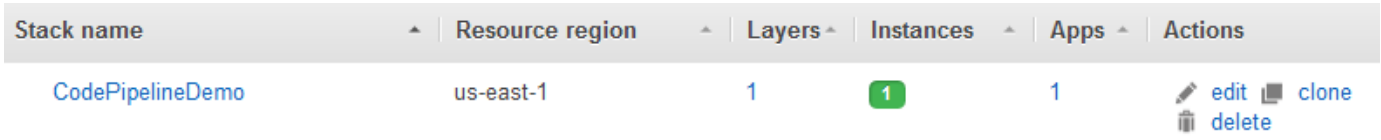
If creating IAM resources is acceptable, select the **I acknowledge that this template might cause AWS CloudFormation to create IAM resources** check box, and then choose **Create**. If creating IAM resources is not acceptable, you cannot continue with this procedure.

10. On the AWS CloudFormation dashboard, you can view the progress of the creation of the stack. Before you continue to the next step, wait until **CREATE_COMPLETE** is displayed in the **Status** column.



To verify stack creation in OpsWorks Stacks

1. Open the OpsWorks console at https://console.aws.amazon.com/opsworks/.
2. On the OpsWorks Stacks dashboard, view the stack you created.



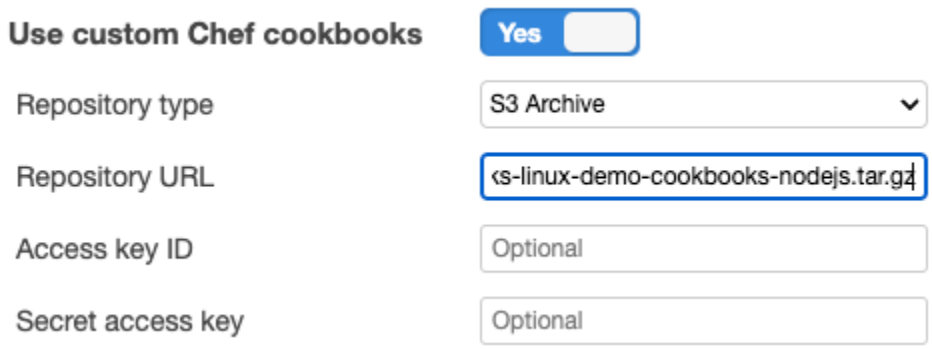
3. Open the stack, and view the layer and instance. Observe that the layer and instance were created with the names and other metadata provided in the AWS CloudFormation template. You are ready to configure your stack and layer to use custom Chef cookbooks and recipes.

Step 2: Configure your stack and layer to use custom cookbooks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Chef 12 stacks in OpsWorks Stacks require your own or community-created cookbooks to build custom application layers. For this walkthrough, you can point to a repository that contains a set of Chef cookbooks and Chef recipes. These recipes install the Node.js package and its dependencies on your instance. You will use other Chef recipes to deploy the Node.js app that you will prepare in Step 4: Add your app to OpsWorks Stacks. The Chef recipe that you specify in this step runs every time a new version of your application is deployed by CodePipeline.

1. In the OpsWorks Stacks console, open the stack that you created in Step 1: Create a stack, layer, and an instance in OpsWorks Stacks. Choose **Stack Settings**, and then choose **Edit**.
2. Set **Use custom Chef cookbooks** to **Yes**. This shows related custom cookbook settings.
3. From the **Repository type** drop-down list, choose **S3 Archive**. To work with both CodePipeline and OpsWorks, your cookbook source must be S3.
4. For **Repository URL**, specify **`https://s3.amazonaws.com/opsworks-demo-assets/opsworks-linux-demo-cookbooks-nodejs.tar.gz`**. Your settings should resemble the following.



5. Choose **Save**.
6. In the navigation pane, choose **Layers**.

7. Choose **Settings** for the layer you created in Step 1: Create a stack, layer, and an instance in OpsWorks Stacks.
8. On the **General Settings** tab, be sure that the layer name is **Node.js App Server**, and the layer short name is **app1**. Choose **Recipes**.
9. On the **Recipes** tab, specify **`nodejs_demo`** as the recipe you want to run during the **Deploy** lifecycle event. Choose **Save**.
10. On the **Security** tab, from the **Security groups** drop-down list, choose the **AWS-OpsWorks-Webapp** security group.
11. Choose **Save**.

Step 3: Upload app code to an Amazon S3 bucket

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Because you must provide a link to your code repository as part of pipeline setup, have the code repository ready before you create your pipeline. In this walkthrough, you upload a Node.js app to an Amazon S3 bucket.

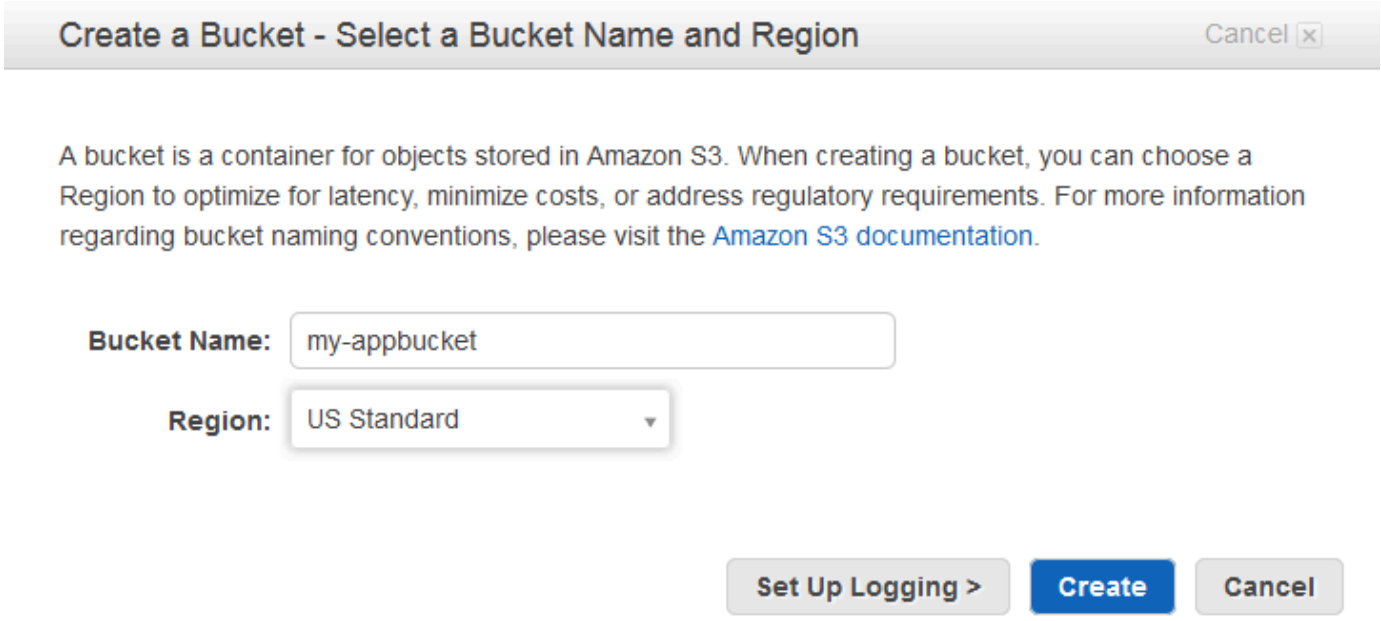
Although CodePipeline can use code directly from GitHub or CodeCommit as sources, this walkthrough demonstrates how to use an Amazon S3 bucket. In this walkthrough, you upload the sample Node.js app to your own Amazon S3 bucket, so you can make changes to the app. The Amazon S3 bucket that you create in this step enables CodePipeline to detect changes to the app code and deploy the changed app automatically. If you wish, you can use an existing bucket. Be sure the bucket meets the criteria described in Simple Pipeline Walkthrough (Amazon S3 Bucket) in the CodePipeline documentation.

> **Important**
>
> The Amazon S3 bucket must be in the same region in which you will later create your pipeline. At this time, CodePipeline supports the OpsWorks Stacks provider in the US East

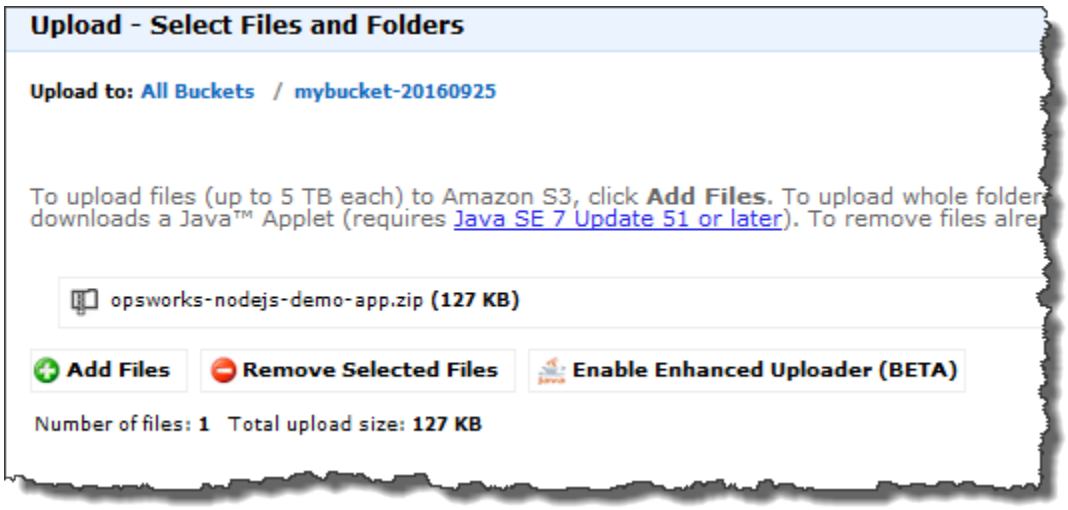
(N. Virginia) Region (us-east-1) only. All resources in this walkthrough should be created in the US East (N. Virginia) Region. The bucket must also be versioned because CodePipeline requires a versioned source. For more information, see Using Versioning.

To upload your app to an Amazon S3 bucket

1. Download the ZIP file of the OpsWorks Stacks sample, Node.js app, and save it to a convenient location on your local computer.
2. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
3. Choose **Create Bucket**.
4. On the **Create a Bucket - Select a Bucket Name and Region** page, for **Bucket Name**, type a unique name for your bucket. Bucket names must be unique across all AWS accounts, not just in your own account. This walkthrough uses the name **`my-appbucket`**, but you can use `my-appbucket-`*`yearmonthday`* to make your bucket name unique. From the **Region** drop-down list, choose **US Standard**, and then choose **Create**. **US Standard** is equivalent to `us-east-1`.



5. Choose the bucket you created from the **All Buckets** list.
6. On the bucket page, choose **Upload**.
7. On the **Upload - Select Files and Folders** page, choose **Add files**. Browse for the ZIP file you saved in step 1, choose **Open**, and then choose **Start Upload**.



8. After the upload is complete, select the ZIP file from the list of files in your bucket, and then choose **Properties**.
9. In the **Properties** pane, copy the link to your ZIP file, and make a note of the link. You will need the bucket name and the ZIP file name portion of this link to create your pipeline.

Step 4: Add your app to OpsWorks Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Before you create a pipeline in CodePipeline, add the Node.js test app to OpsWorks Stacks. When you create the pipeline, you will need to select the app that you've added to OpsWorks Stacks.

Have the Amazon S3 bucket link from step 9 of the preceding procedure ready. You will need the link to the bucket in which you stored your test app to complete this procedure.

To add an app to OpsWorks Stacks

1. In the OpsWorks Stacks console, open **CodePipelineDemo**, and in the navigation pane, choose **Apps**.

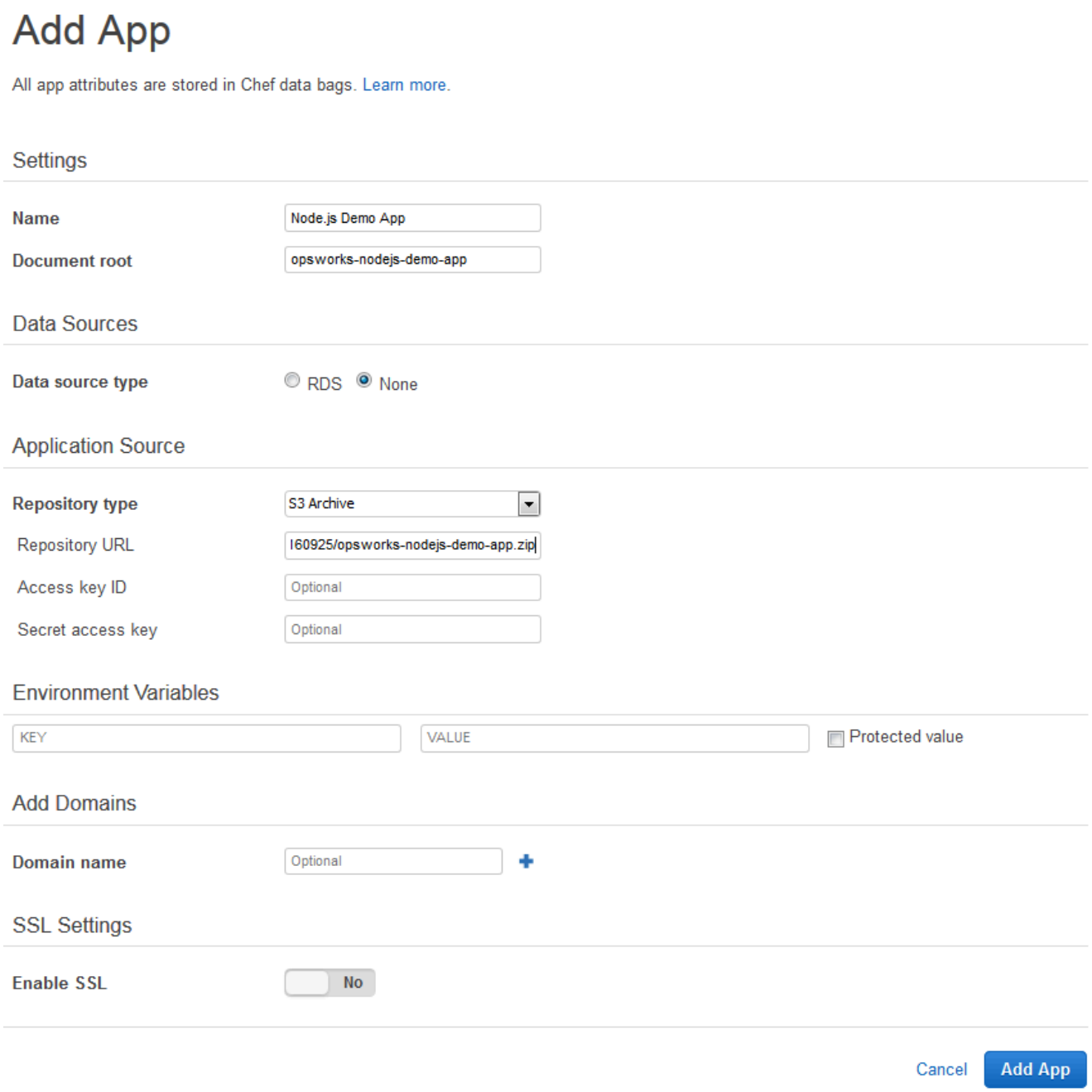
2. Choose **Add app**.
3. On the **Add App** page, provide the following information:

   a. Specify a name for your app. This walkthrough uses the name `Node.js Demo App`.

   b. For **Data source type**, choose **None**. This app does not require an external database or data source.

   c. In the **Repository type** drop-down list, choose **S3 Archive**.

   d. In the **Repository URL** string box, paste the URL that you copied in step 9 of Step 3: Upload app code to an Amazon S3 bucket. Your form should be similar to the following:



4. You do not need to change any other settings in this form. Choose **Add App**.
5. When the **Node.js Demo App** app appears in the list on the **Apps** page, continue to the next procedure, Step 5: Create a pipeline in CodePipeline.

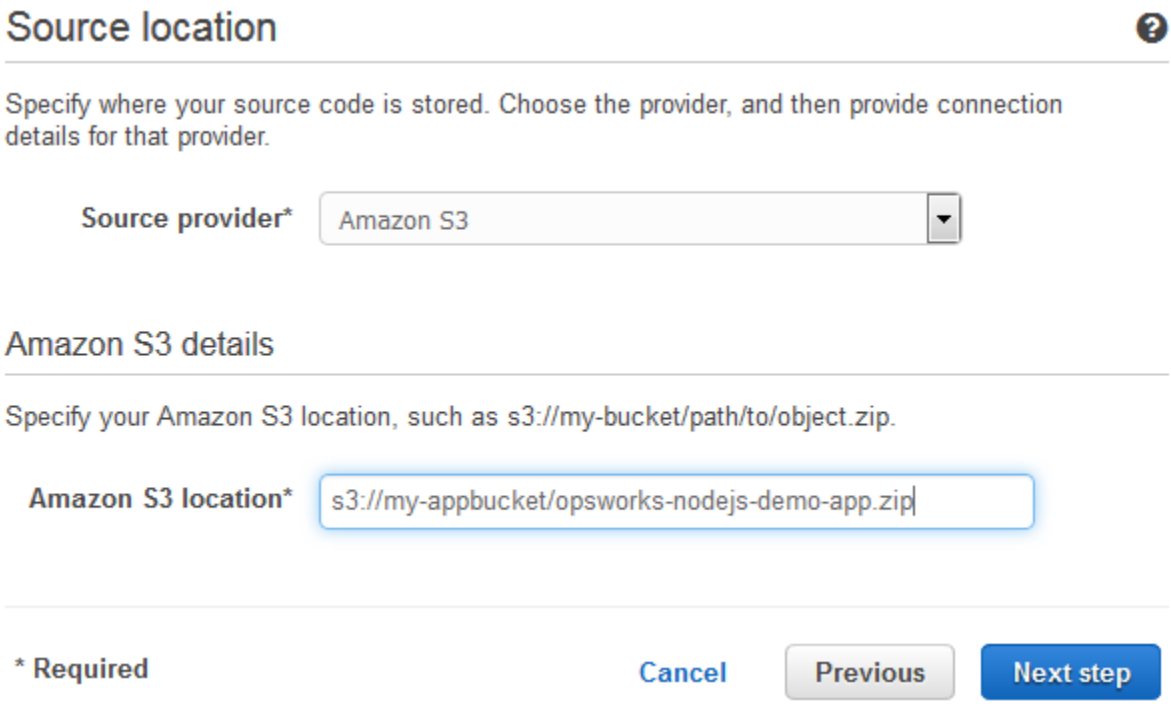
Step 5: Create a pipeline in CodePipeline

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

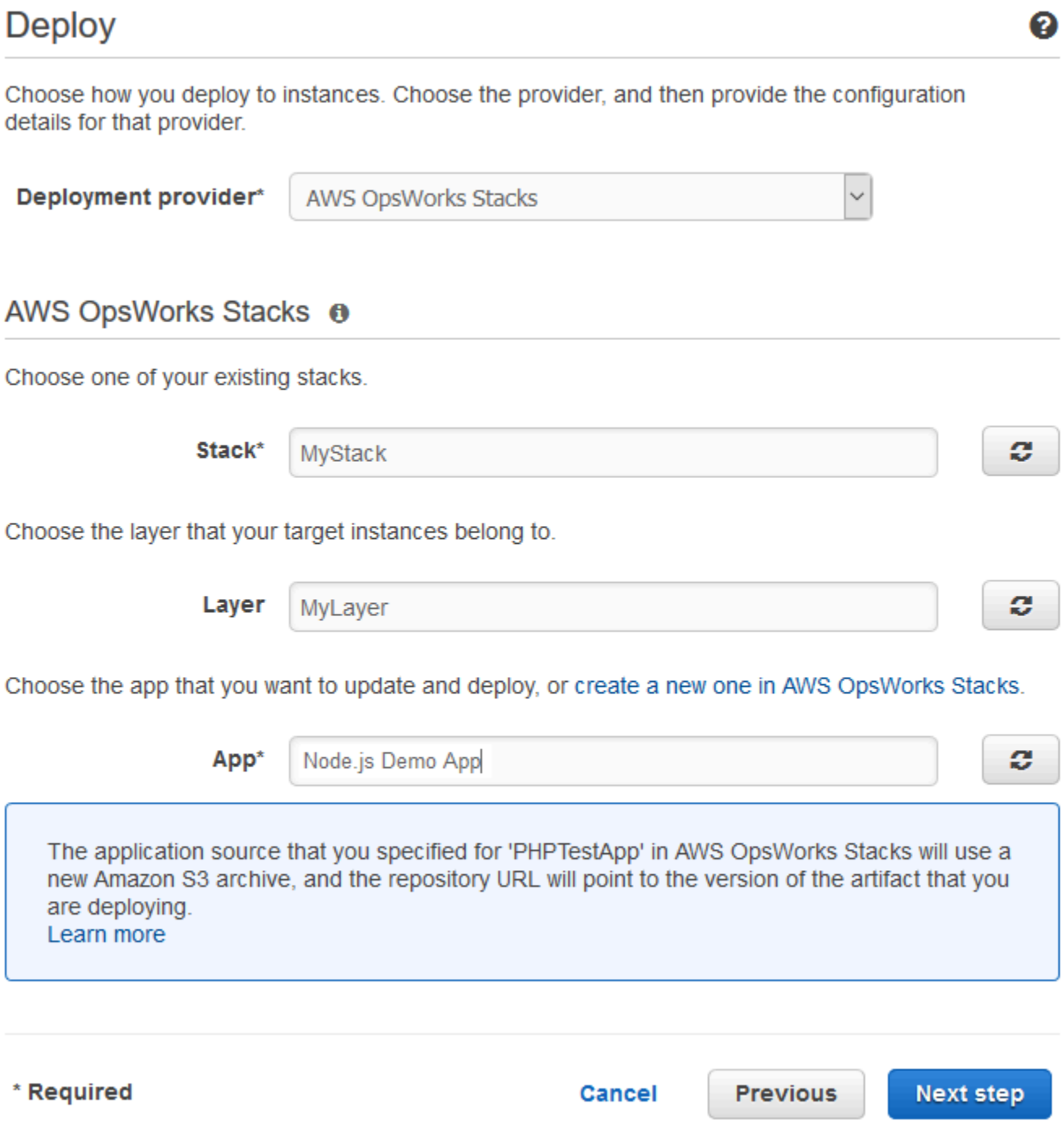
After you have a stack with a layer and at least one instance configured in OpsWorks Stacks, create a pipeline in CodePipeline with OpsWorks Stacks as the provider to deploy apps or Chef cookbooks to your OpsWorks Stacks resources.

To create a pipeline

1. Open the CodePipeline console at https://console.aws.amazon.com/codepipeline/.
2. Choose **Create pipeline**.
3. On the **Getting started with CodePipeline** page, type **`MyOpsWorksPipeline`**, or any other pipeline name that is unique to your account, and then choose **Next step**.
4. On the **Source Location** page, select **Amazon S3** from the **Source provider** drop-down list.
5. In the **Amazon S3 details** area, type your Amazon S3 bucket path, in the format **`s3://bucket-name/file name`**. Refer to the link that you noted in step 9 of Step 3: Upload app code to an Amazon S3 bucket. In this walkthrough, the path is `s3://my-appbucket/opsworks-nodejs-demo-app.zip`. Choose **Next step**.



6. On the **Build** page, choose **No Build** from the drop-down list, and then choose **Next step**.
7. On the **Deploy** page, choose **OpsWorks Stacks** as the deployment provider.



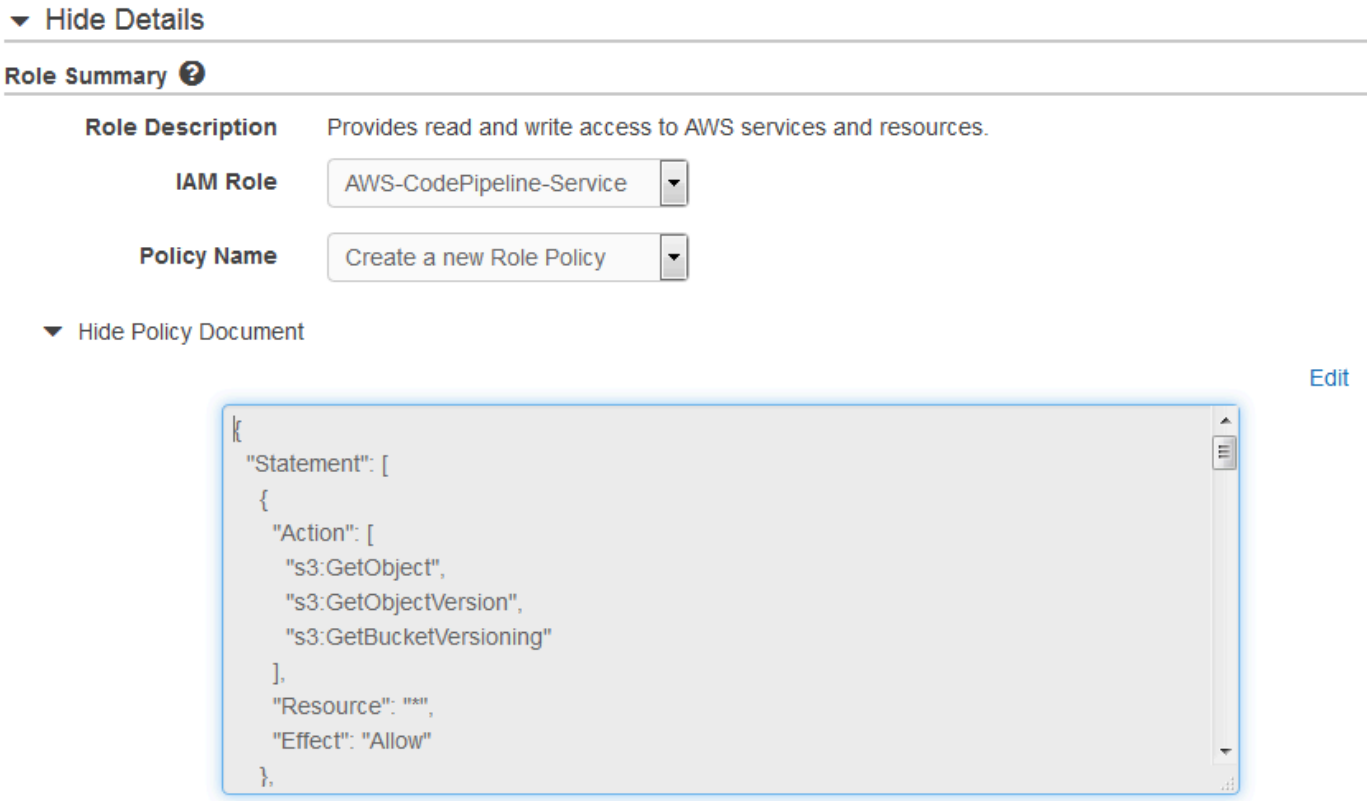
8. In the **Stack** field, type `CodePipelineDemo`, or the name of the stack that you created in Step 1: Create a stack, layer, and an instance in OpsWorks Stacks.
9. In the **Layer** field, type `Node.js App Server`, or the name of the layer that you created in Step 1: Create a stack, layer, and an instance in OpsWorks Stacks.

10. In the **App** field, select the app that you uploaded to Amazon S3 in Step 3: Upload app code to an Amazon S3 bucket, and then choose **Next step**.
11. On the **AWS Service Role** page, choose **Create Role**.

    A new window opens with an IAM console page that describes the role that will be created for you, `AWS-CodePipeline-Service`. From the **Policy name** drop-down list, choose **Create new policy**. Be sure the policy document has the following content. Choose **Edit** to change the policy document, if required.

```
{
    "Statement": [
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:GetBucketVersioning"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": "opsworks:*",
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

    When you are finished making changes to the policy document, choose **Allow**. Your changes will be displayed in the IAM console.



> **Note**
>
> If role creation fails, it might be because you already have an IAM role named **AWS-CodePipeline-Service**. If you have been using the **AWS-CodePipeline-Service** role before May 2016, the role might not have permissions to use OpsWorks Stacks as a deployment provider. In this case, you must update the policy statement as shown in this step. If you see an error message, go back to the beginning of this step, and choose **Use existing role** instead of **Create role**. If you use an existing role, the role should have a policy attached that includes the permissions shown in this step. For more information about the service role and its policy statement, see Edit a Policy for an IAM Service Role.

12. If the role creation process is successful, the IAM page will close, and you will be returned to the **AWS Service Role** page. Choose **Next step**.
13. On the **Review your pipeline** page, verify the choices shown on the page, and then choose **Create pipeline**.
14. When your pipeline is ready, it should start locating your source code and deploying your app to your stack automatically. This process can take several minutes.

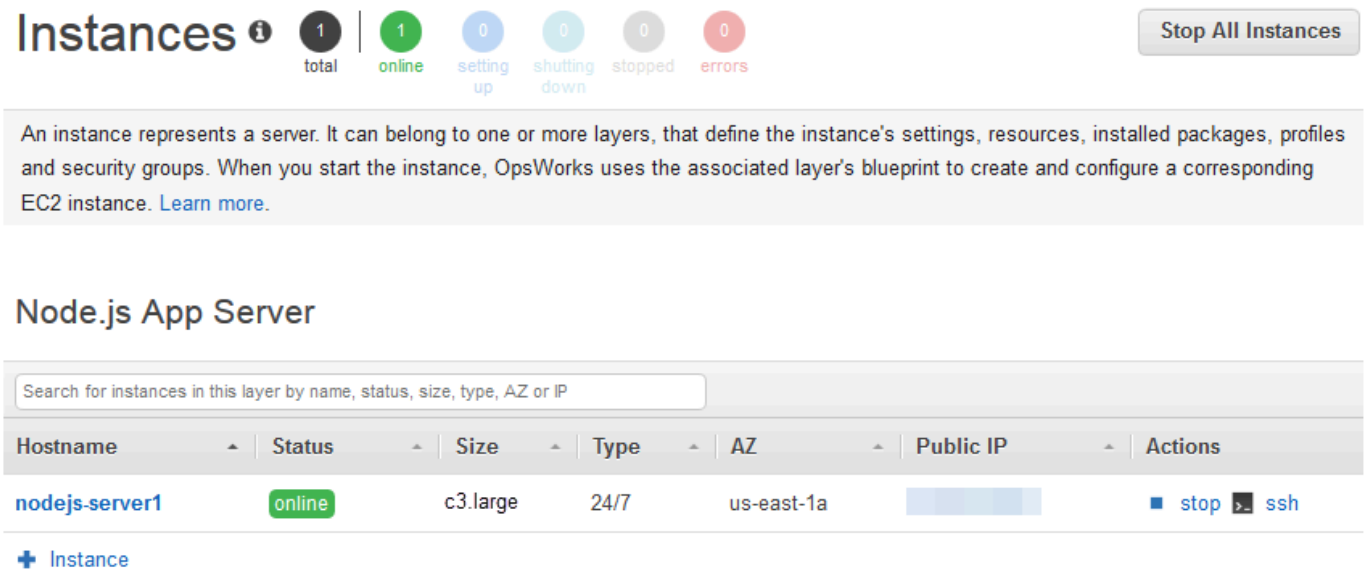
Step 6: Verifying the app deployment in OpsWorks Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

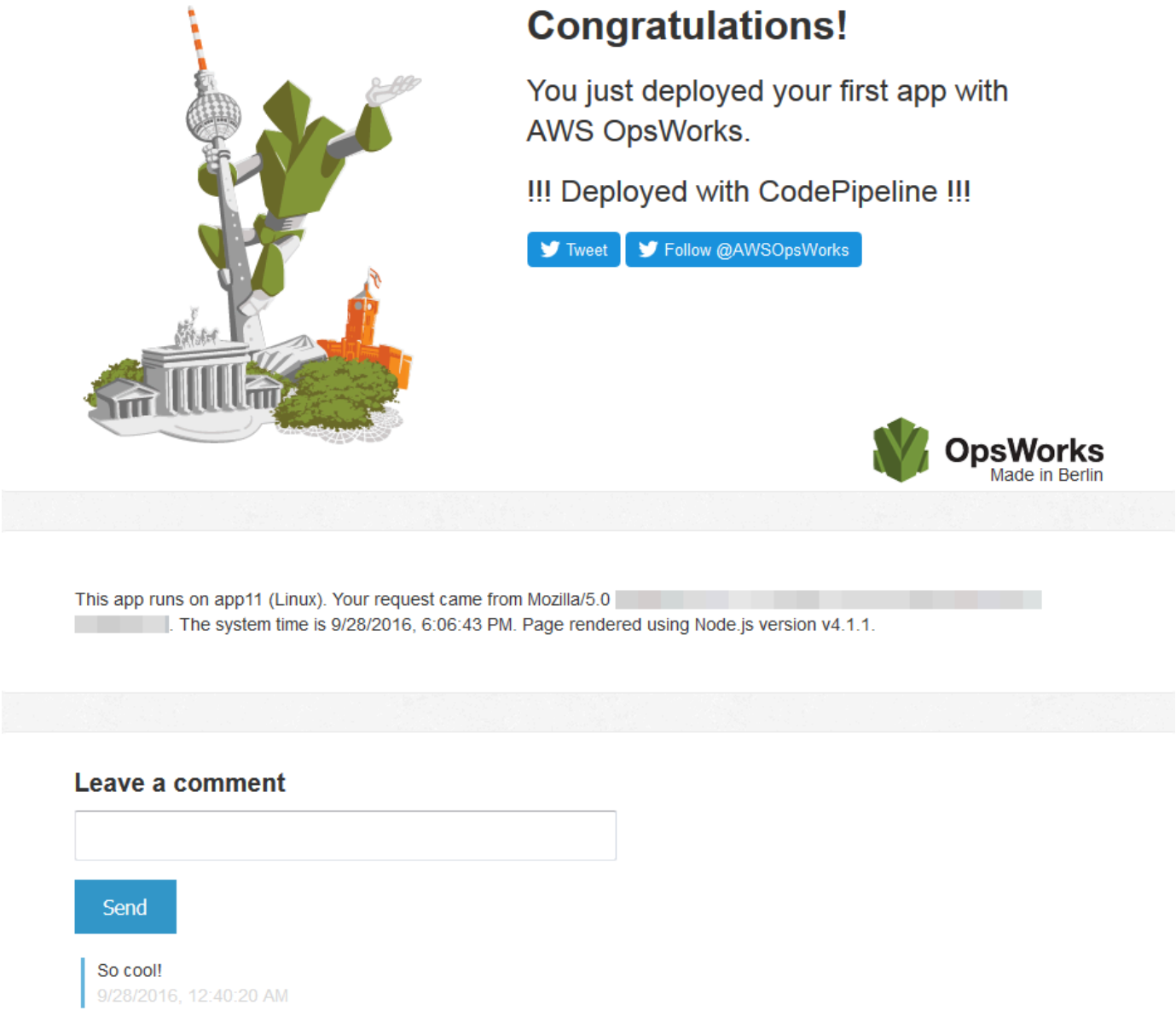
To verify that CodePipeline deployed the Node.js app to your stack, sign in to the instance you created in Step 1: Create a stack, layer, and an instance in OpsWorks Stacks. You should be able to see and use the Node.js web app.

To verify the app deployment in your OpsWorks Stacks instance

1. Open the OpsWorks console at https://console.aws.amazon.com/opsworks/.
2. On the OpsWorks Stacks dashboard, choose **CodePipelineDemo**, and then choose **Node.js App Server**.
3. In the navigation pane, choose **Instances**, and then choose the public IP address of the instance that you created to view the web app.



The app will be displayed in a new browser tab.



Step 7 (Optional): Update the app code to see CodePipeline redeploy your app automatically

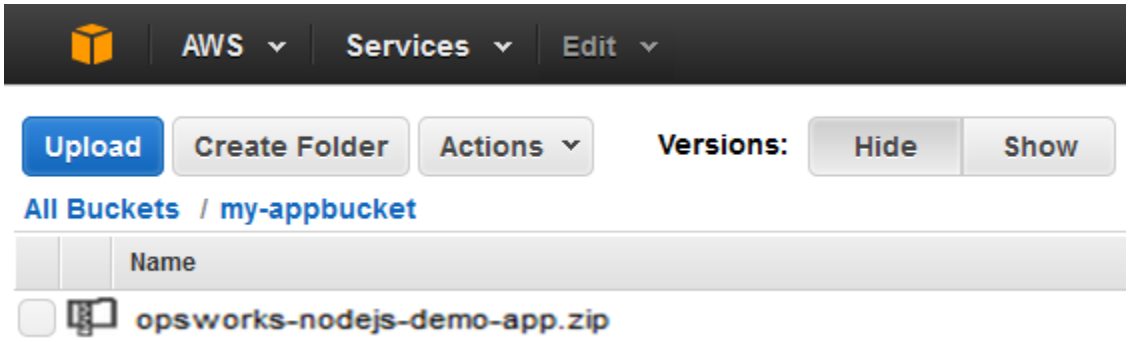
> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

When you make changes to code in apps or cookbooks that you have deployed by using CodePipeline, the updated artifacts will be deployed automatically by CodePipeline to your target instances (in this case, to a target OpsWorks Stacks stack). This section shows you the automatic redeployment when you update the code in your sample Node.js app. If you still have the app code for this walkthrough stored locally, and no one else has made changes to the code since you started the walkthrough, you can skip steps 1-4 of this procedure.

To edit the code in the sample app

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Open the bucket in which you are storing your sample Node.js app.

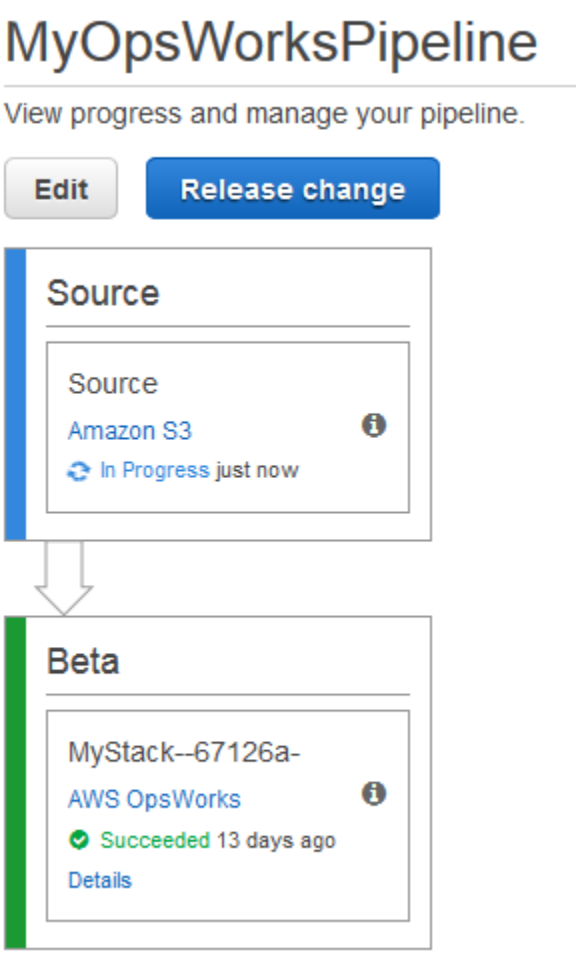


3. Select the ZIP file that contains the app. On the **Actions** menu, choose **Download**.
4. In the dialog box, open the context (right-click) menu, choose **Download**, and then save the ZIP file to a convenient location. Choose **OK**.
5. Extract the contents of the ZIP file to a convenient location. You might need to change permissions on the extracted folder and its subfolders and contents to allow editing. In the `opsworks-nodejs-demo-app\views` folder, open the `header.html` file for editing.
6. Search for the phrase, `You just deployed your first app with`. Replace the word `deployed` with `updated`. On the next line, change `OpsWorks.` to `OpsWorks and AWS CodePipeline.` Do not edit anything but the text.

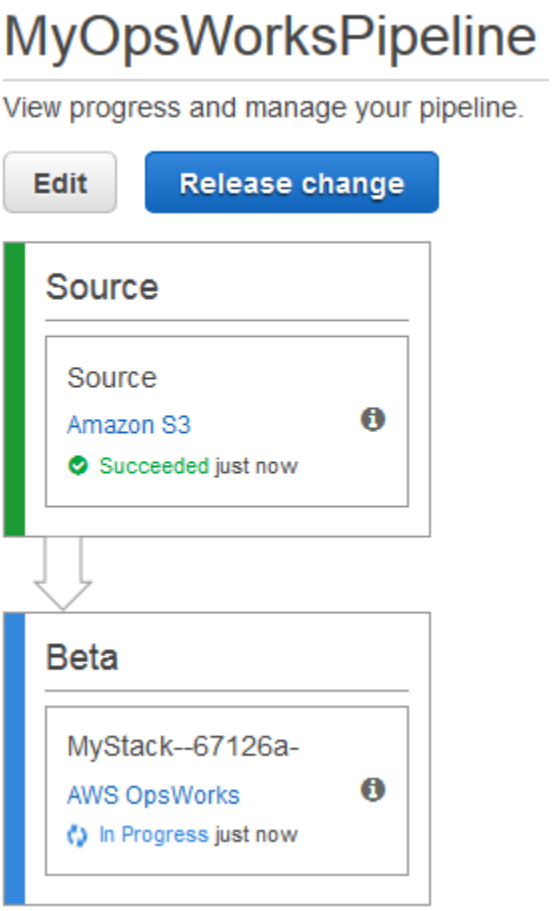
```
<div id="main" role="main">LF
  <div class="container">LF
    <div class="hero-unit">LF
      <div class="robot">LF
        <h1>Congratulations!</h1>LF
        <h2>LF
          You just updated your first app with<br/>LF
          AWS OpsWorks and AWS CodePipeline.LF
        </h2>LF
```

7. Save and close the `header.html` file.
8. Zip the `opsworks-nodejs-demo-app` folder, and save the ZIP file to a convenient location. Do not change the name of the ZIP file.
9. Upload the new ZIP file to your Amazon S3 bucket. In this walkthrough, the name of the bucket is `my-appbucket`.
10. Open the CodePipeline console, and open your OpsWorks Stacks pipeline (**MyOpsWorksPipeline**). Choose **Release Change**.

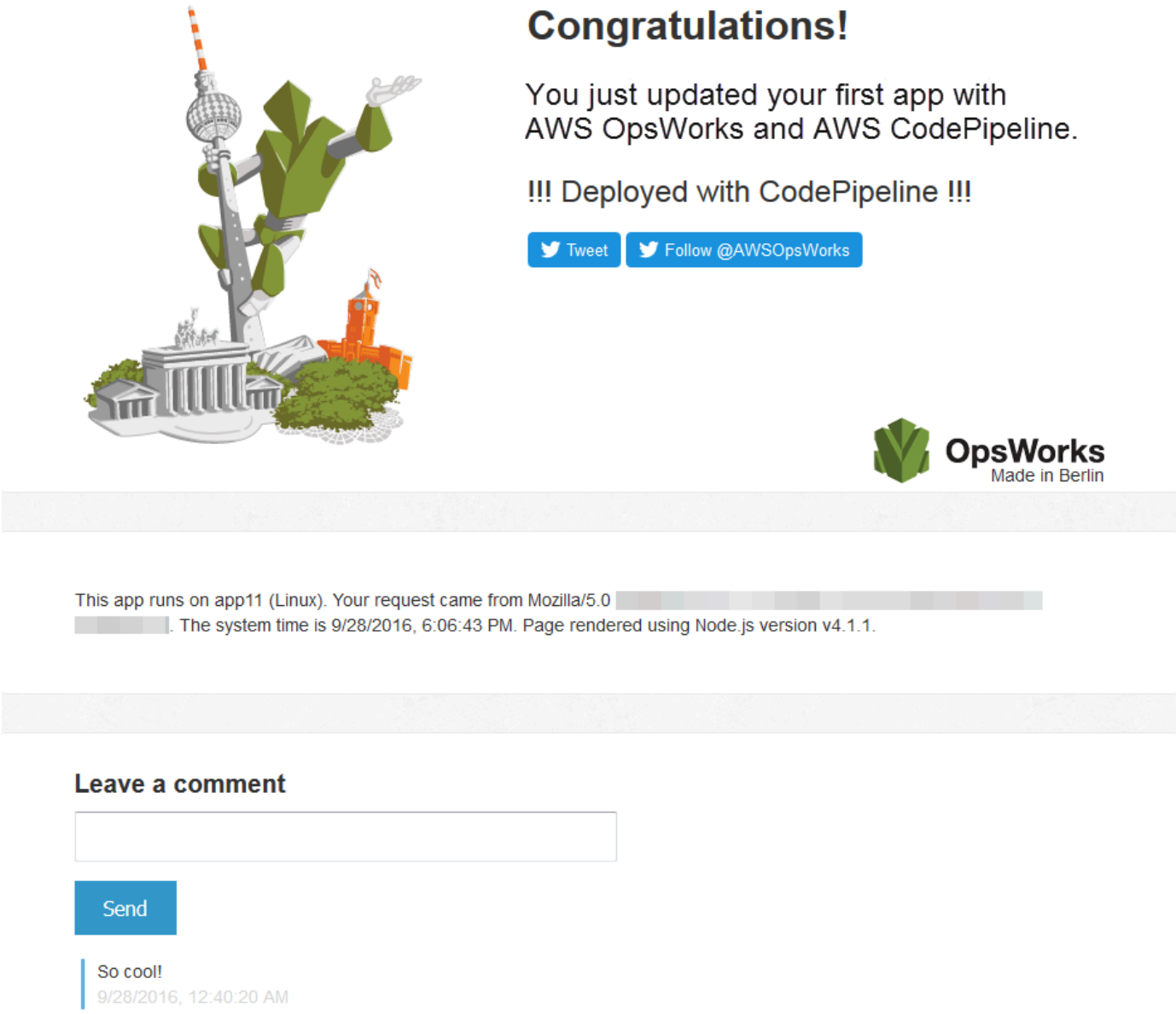
    (You can wait for CodePipeline to detect the code change from the updated version of the app in your Amazon S3 bucket. To save you time, this walkthrough instructs you to simply choose **Release Change**.)
11. Observe as CodePipeline runs through the stages of the pipeline. First, CodePipeline detects changes to the source artifact.



CodePipeline pushes the updated code to your stack in OpsWorks Stacks.



12. When both stages of the pipeline have been successfully completed, open your stack in OpsWorks Stacks.
13. On the stack properties page, choose **Instances**.
14. In the **Public IP** column, choose the public IP address of your instance to view the updated app's text.



Step 8 (Optional): Clean up resources

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

To help prevent unwanted charges to your AWS account, you can delete the AWS resources that you used for this walkthrough. These AWS resources include the OpsWorks Stacks stack, the IAM role and instance profile, and the pipeline that you created in CodePipeline. However, you might want to keep using these AWS resources as you continue to learn more about OpsWorks Stacks and CodePipeline. If you want to keep these resources, you have finished this walkthrough.

To delete the app from the stack

Because you did not create or apply the app as part of your AWS CloudFormation template, delete the Node.js test app before you delete the stack in AWS CloudFormation.

1. In the OpsWorks Stacks console, in the service navigation pane, choose **Apps**.
2. On the **Apps** page, select **Node.js Demo App**, and then in **Actions**, choose **delete**. When you are prompted to confirm, choose **Delete**. OpsWorks Stacks will delete the app.

To delete the stack

Because you created the stack by running an AWS CloudFormation template, you can delete the stack, including the layer, instance, instance profile, and security group that the template created, in the AWS CloudFormation console.

1. Open the AWS CloudFormation console.
2. In the AWS CloudFormation console dashboard, select the stack you created. On the **Actions** menu, choose **Delete Stack**. When you are prompted to confirm, choose **Yes, Delete**.
3. Wait for **DELETE_COMPLETE** to appear in the **Status** column for the stack.

To delete the pipeline

1. Open the CodePipeline console.
2. In the CodePipeline dashboard, choose the pipeline you created for this walkthrough.
3. On the pipeline page, choose **Edit**.
4. On the **Edit** page, choose **Delete**. When you are prompted to confirm, choose **Delete**.

AWS CodePipeline with OpsWorks Stacks - Chef 11 Stacks

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

AWS CodePipeline lets you create continuous delivery pipelines that track code changes from sources such as CodeCommit, Amazon Simple Storage Service (Amazon S3), or GitHub. The example in this topic describes how to create and use a simple pipeline from CodePipeline as a deployment tool for code that you run on OpsWorks Stacks layers. In this example, you create a pipeline for a simple PHP app, and then instruct OpsWorks Stacks to run the app on all of the instances in a layer in a Chef 11.10 stack (in this case, a single instance).

Note

This topic describes how to use a pipeline to run and update an app on a Chef 11.10 stack. For information about how to use a pipeline to run and update an app on a Chef 12 stack, see AWS CodePipeline with OpsWorks Stacks - Chef 12 Stacks. Content delivered to Amazon S3 buckets might contain customer content. For more information about removing sensitive data, see How Do I Empty an S3 Bucket? or How Do I Delete an S3 Bucket?.

Topics

- Prerequisites
- Other Supported Scenarios
- Step 1: Create a stack, layer, and an instance in OpsWorks Stacks
- Step 2: Upload app code to an Amazon S3 bucket
- Step 3: Add your app to OpsWorks Stacks
- Step 4: Create a pipeline in CodePipeline
- Step 5: Verifying the app deployment in OpsWorks Stacks

- Step 6 (Optional): Update the app code to see CodePipeline redeploy your app automatically
- Step 7 (Optional): Clean up resources

Prerequisites

Before you start this walkthrough, be sure that you have administrator permissions to perform all of the following tasks. You can be a member of a group that has the **AdministratorAccess** policy applied, or you can be a member of a group that has the permissions and policies shown in the following table. As a security best practice, you should belong to a group that has permissions to do the following tasks, instead of assigning required permissions to individual users.

For more information about creating a security group in IAM and assigning permissions to the group, see Creating IAM user groups. For more information about managing OpsWorks Stacks permissions, see Best Practices: Managing Permissions.

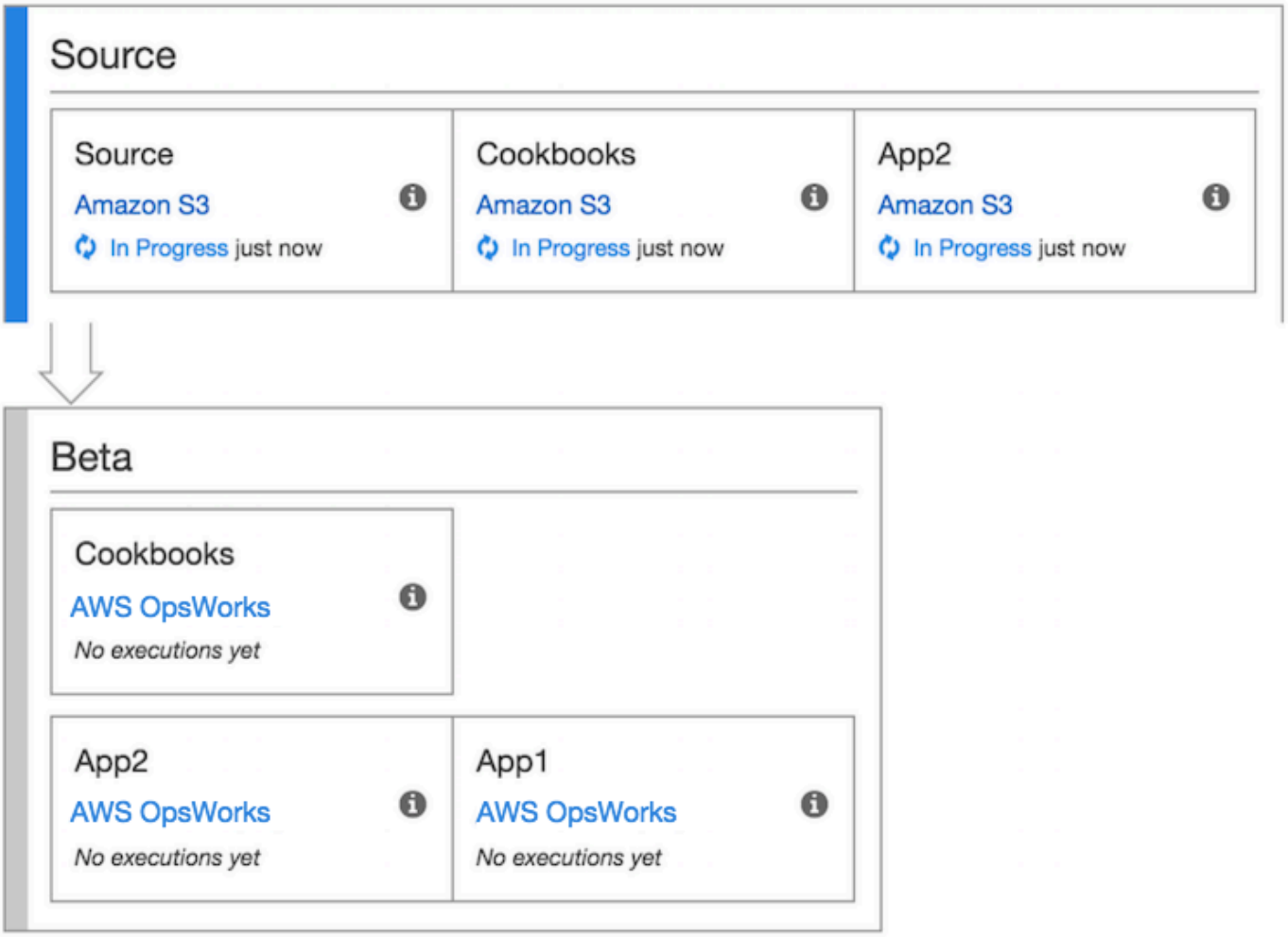
Permissions	Recommended Policy to Attach to Group
Create and edit stacks, layers, and instances in OpsWorks Stacks.	**AWSOpsWorks_FullAccess**
Create, edit, and run templates in AWS CloudFormation.	**AmazonCloudFormationFullAccess**
Create, edit, and access Amazon S3 buckets.	**AmazonS3FullAccess**
Create, edit, and run pipelines in CodePipeline, especially pipelines that use OpsWorks Stacks as the provider.	**AWSCodePipeline_FullAccess**

You must also have an Amazon EC2 key pair. You will be prompted to provide the name of this key pair when you run the AWS CloudFormation template that creates the sample stack, layer, and instance in this walkthrough. For more information about obtaining a key pair in the Amazon EC2 console, see Create a Key Pair in the Amazon EC2 documentation. The key pair should be in the US East (N. Virginia) Region. You can use an existing key pair if you already have one in that region.

Other Supported Scenarios

This walkthrough creates a simple pipeline that includes one **Source** and one **Deploy** stage. However, you can create more complex pipelines that use OpsWorks Stacks as a provider. The following are examples of supported pipelines and scenarios:

- You can edit a pipeline to add a Chef cookbook to the **Source** stage and an associated target for updated cookbooks to the **Deploy** stage. In this case, you add a **Deploy** action that triggers the updating of your cookbooks when you make changes to the source. The updated cookbook is deployed before your app.
- You can create a complex pipeline, with custom cookbooks and multiple apps, and deploy to an OpsWorks Stacks stack. The pipeline tracks changes to both the application and cookbook sources, and redeploys when you have made changes. The following shows an example of a similar, complex pipeline:



For more information about working with CodePipeline, see the CodePipeline documentation.

Step 1: Create a stack, layer, and an instance in OpsWorks Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

To use OpsWorks Stacks as a deployment provider for a pipeline, you must first have a stack, a layer, and at least one instance in the layer. Although you can create a stack in OpsWorks Stacks by following instructions in Getting Started with Linux Stacks or Getting Started with Windows Stacks, to save you time, this example uses an AWS CloudFormation template to create a Linux-based Chef 11.10 stack, layer, and instance. The instance created by this template runs Amazon Linux 2016.03, and has an instance type of `c3.large`.

> **Important**
>
> The AWS CloudFormation template must be stored and run in the same region as the Amazon S3 bucket to which you later upload your app and the same region in which you later create your pipeline in CodePipeline. At this time, CodePipeline supports the OpsWorks Stacks provider in the US East (N. Virginia) Region (us-east-1) only. All resources in this walkthrough should be created in the US East (N. Virginia) Region.
> If stack creation fails, you might be approaching the maximum allowed number of IAM roles for your account. The stack creation can also fail if your account cannot launch instances with a `c3.large` instance type. For example, if you are using the AWS Free Tier, you might receive an error such as `Root device type: must be included in EBS`. If your account has limitations on the instance types that you are allowed to create, such as limitations imposed by the AWS Free Tier, try changing the value of the `InstanceType` parameter in the template's instance block to an instance type that your account can use.

To create a stack, layer, and instance using AWS CloudFormation

1. Copy the following AWS CloudFormation template into a new plain-text document. Save the file to a convenient location on your local computer, and name it **NewOpsWorksStack.template**, or another name that is convenient for you.

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Mappings": {
    "Region2Principal": {
      "us-east-1": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "us-west-2": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "us-west-1": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "eu-west-1": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "ap-southeast-1": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "ap-northeast-1": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "ap-northeast-2": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "ap-southeast-2": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "sa-east-1": {
```

```
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      },
      "cn-north-1": {
        "EC2Principal": "ec2.amazonaws.com.cn",
        "OpsWorksPrincipal": "opsworks.amazonaws.com.cn"
      },
      "eu-central-1": {
        "EC2Principal": "ec2.amazonaws.com",
        "OpsWorksPrincipal": "opsworks.amazonaws.com"
      }
    }
 },
 "Parameters": {
   "EC2KeyPairName": {
  "Type": "String",
  "Description": "The name of an existing EC2 key pair that allows you to use SSH
to connect to the OpsWorks instance."
 }
 },
 "Resources": {
"CPOpsDeploySecGroup": {
  "Type": "AWS::EC2::SecurityGroup",
  "Properties": {
    "GroupDescription" : "Lets you manage OpsWorks instances deployed to by
CodePipeline"
  }
},
"CPOpsDeploySecGroupIngressHTTP": {
  "Type": "AWS::EC2::SecurityGroupIngress",
  "Properties" : {
    "IpProtocol" : "tcp",
       "FromPort" : "80",
       "ToPort" : "80",
       "CidrIp" : "0.0.0.0/0",
 "GroupId": {
   "Fn::GetAtt": [
     "CPOpsDeploySecGroup", "GroupId"
   ]
 }
     }
},
"CPOpsDeploySecGroupIngressSSH": {
  "Type": "AWS::EC2::SecurityGroupIngress",
```

```
   "Properties" : {
     "IpProtocol" : "tcp",
       "FromPort" : "22",
       "ToPort" : "22",
       "CidrIp" : "0.0.0.0/0",
 "GroupId": {
   "Fn::GetAtt": [
     "CPOpsDeploySecGroup", "GroupId"
   ]
 }
  }
},
"MyStack": {
     "Type": "AWS::OpsWorks::Stack",
     "Properties": {
       "Name": {
         "Ref": "AWS::StackName"
       },
       "ServiceRoleArn": {
         "Fn::GetAtt": [
           "OpsWorksServiceRole",
           "Arn"
         ]
       },
 "ConfigurationManager" : { "Name": "Chef","Version": "11.10" },
 "DefaultOs": "Amazon Linux 2016.03",
       "DefaultInstanceProfileArn": {
         "Fn::GetAtt": [
           "OpsWorksInstanceProfile",
           "Arn"
         ]
       }
     }
   },
   "MyLayer": {
     "Type": "AWS::OpsWorks::Layer",
     "Properties": {
       "StackId": {
         "Ref": "MyStack"
       },
       "Name": "MyLayer",
       "Type": "php-app",
 "Shortname": "mylayer",
       "EnableAutoHealing": "true",
```

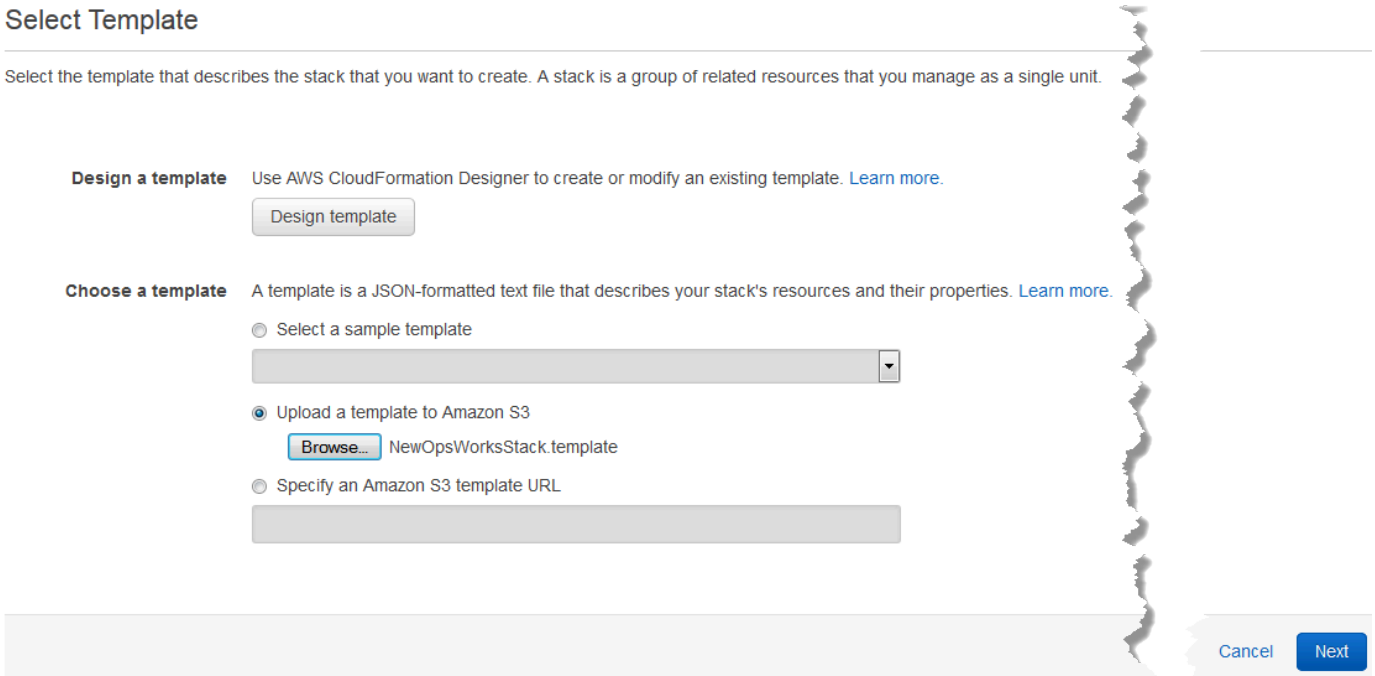
```
        "AutoAssignElasticIps": "false",
        "AutoAssignPublicIps": "true",
    "CustomSecurityGroupIds": [
      {
        "Fn::GetAtt": [
                "CPOpsDeploySecGroup", "GroupId"
        ]
            }
        ]
      },
      "DependsOn": [
        "MyStack",
        "CPOpsDeploySecGroup"
      ]
    },
    "OpsWorksServiceRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  {
                    "Fn::FindInMap": [
                      "Region2Principal",
                      {
                        "Ref": "AWS::Region"
                      },
                      "OpsWorksPrincipal"
                    ]
                  }
                ]
              },
              "Action": [
                "sts:AssumeRole"
              ]
            }
          ]
        },
        "Path": "/",
        "Policies": [
          {
```

```
            "PolicyName": "opsworks-service",
            "PolicyDocument": {
              "Statement": [
                {
                  "Effect": "Allow",
                  "Action": [
                    "ec2:*",
                    "iam:PassRole",
                    "cloudwatch:GetMetricStatistics",
                    "elasticloadbalancing:*"
                  ],
                  "Resource": "*"
                }
              ]
            }
          }
        ]
      }
    },
    "OpsWorksInstanceProfile": {
      "Type": "AWS::IAM::InstanceProfile",
      "Properties": {
        "Path": "/",
        "Roles": [
          {
            "Ref": "OpsWorksInstanceRole"
          }
        ]
      }
    },
    "OpsWorksInstanceRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  {
                    "Fn::FindInMap": [
                      "Region2Principal",
                      {
                        "Ref": "AWS::Region"
```

```
                    },
                    "EC2Principal"
                  ]
                }
              ]
            },
            "Action": [
              "sts:AssumeRole"
            ]
          }
        ]
      },
      "Path": "/",
  "Policies": [
        {
          "PolicyName": "s3-get",
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
              {
                "Effect": "Allow",
                "Action": [
                  "s3:GetObject"
                ],
                "Resource": "*"
              }
            ]
          }
        }
      ]
    }
  },
  "myinstance": {
    "Type": "AWS::OpsWorks::Instance",
    "Properties": {
      "LayerIds": [
        {
          "Ref": "MyLayer"
        }
      ],
      "StackId": {
        "Ref": "MyStack"
      },
      "InstanceType": "c3.large",
```

```
        "SshKeyName": {
    "Ref": "EC2KeyPairName"
  }
      }
    }
  },
  "Outputs": {
    "StackId": {
      "Description": "Stack ID for the newly created AWS OpsWorks stack",
      "Value": {
        "Ref": "MyStack"
      }
    }
  }
}
```

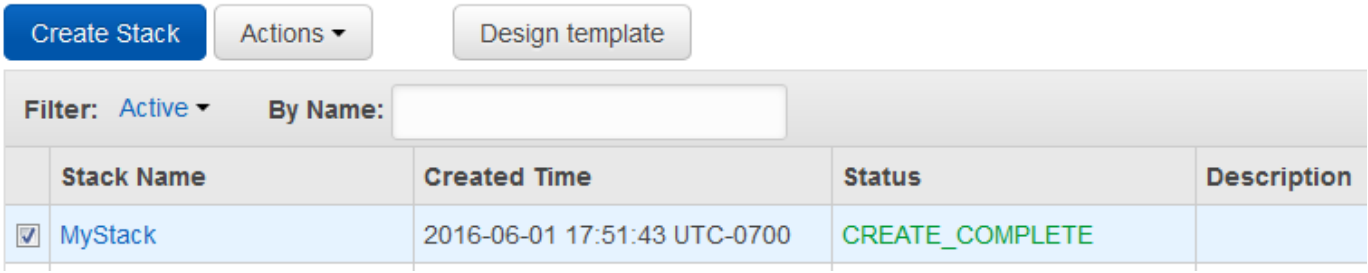
2. Sign in to the AWS Management Console and open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.
3. On the AWS CloudFormation home page, choose **Create stack**.
4. On the **Select Template** page, in the **Choose a template** area, choose **Upload a template to Amazon S3**, and then choose **Browse**.
5. Browse to the AWS CloudFormation template that you saved in step 1, and then choose **Open**. On the **Select Template** page, choose **Next**.



6. On the **Specify Details** page, name the stack **MyStack**, or any stack name that is unique to your account. If you choose a different name for your stack, change the stack name throughout this walkthrough.
7. In the **Parameters** area, provide the name of an EC2 key pair that you want to use to access your OpsWorks Stacks instance after it has been created. Choose **Next**.
8. On the **Options** page, choose **Next**. (Settings on this page are not required for this walkthrough.)
9. The AWS CloudFormation template that you use in this walkthrough creates IAM roles, an instance profile, and an instance.

   > **Important**
   >
   > Before you choose **Create**, choose **Cost** to estimate charges you might incur from AWS for creating resources with this template.

   If creating IAM resources is acceptable, select the **I acknowledge that this template might cause AWS CloudFormation to create IAM resources** check box, and then choose **Create**. If creating IAM resources is not acceptable, you cannot continue with this procedure.
10. On the AWS CloudFormation dashboard, you can view the progress of the creation of the stack. Before you continue to the next step, wait until **CREATE_COMPLETE** is displayed in the **Status** column.



To verify stack creation in OpsWorks Stacks

1. Open the OpsWorks console at https://console.aws.amazon.com/opsworks/.
2. On the OpsWorks Stacks dashboard, view the stack you created.



3. Open the stack, and view the layer and instance. Observe that the layer and instance were created with the names and other metadata provided in the AWS CloudFormation template. You are ready to upload your app to an Amazon S3 bucket.

Step 2: Upload app code to an Amazon S3 bucket

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Because you must provide a link to your code repository as part of pipeline setup, have the code repository ready before you create your pipeline. In this walkthrough, you upload a PHP app to an Amazon S3 bucket.

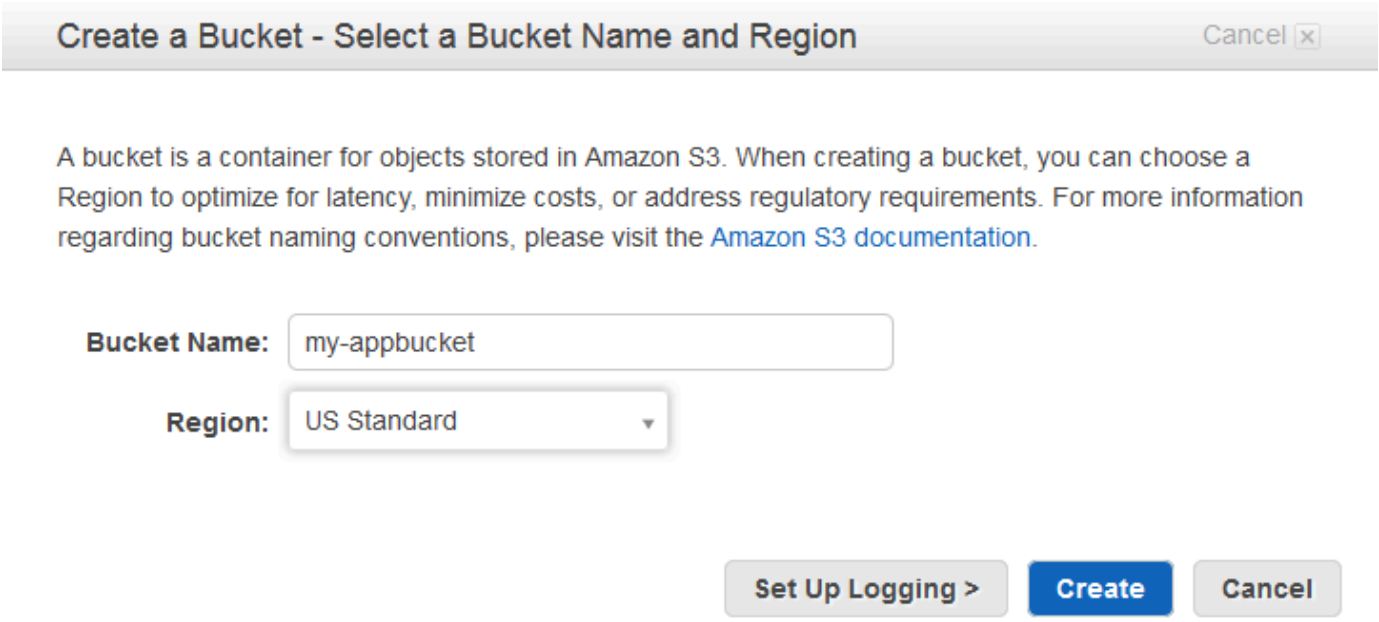
Although CodePipeline can use code directly from GitHub or CodeCommit as sources, this walkthrough demonstrates how to use an Amazon S3 bucket. The Amazon S3 bucket enables CodePipeline to detect changes to the app code and deploy the changed app automatically. If you wish, you can use an existing bucket. Be sure the bucket meets criteria for CodePipeline as described in Simple Pipeline Walkthrough (Amazon S3 Bucket) in the CodePipeline documentation.

> **Important**
>
> The Amazon S3 bucket must be in the same region in which you later create your pipeline. At this time, CodePipeline supports the OpsWorks Stacks provider in the US East (N. Virginia) Region (us-east-1) only. All resources in this walkthrough should be created in the US East (N. Virginia) Region. The bucket must also be versioned because CodePipeline requires a versioned source. For more information, see Using Versioning.

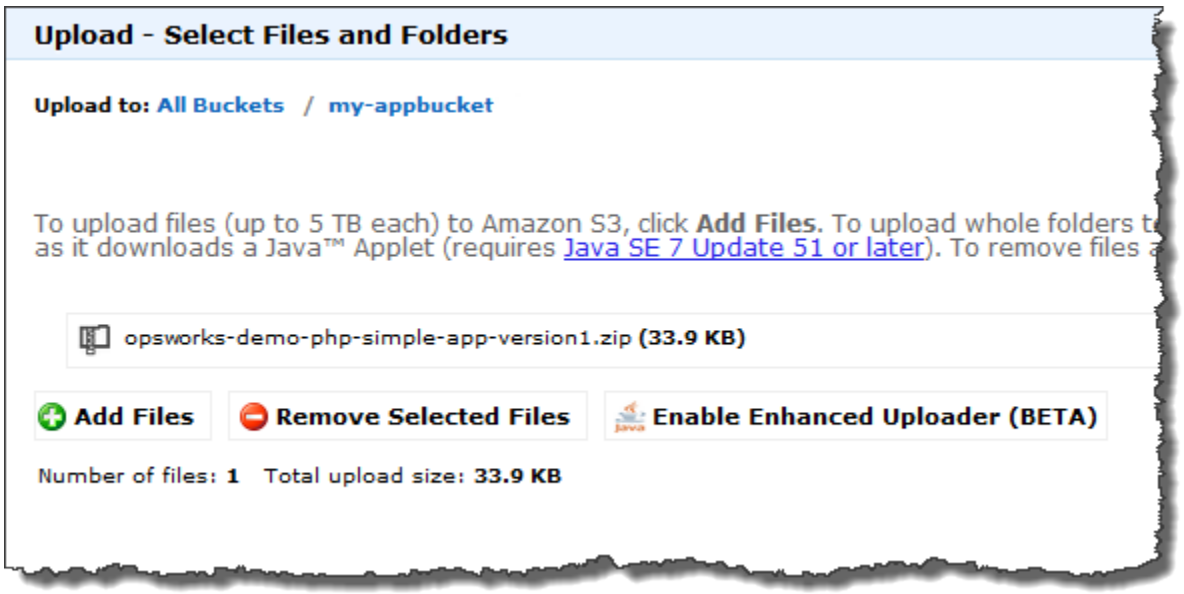
To upload your app to an Amazon S3 bucket

1. From the GitHub website, download a ZIP file of the OpsWorks Stacks sample PHP app, and save it to a convenient location on your local computer.

2. Be sure that `index.php` and the ASSETS folder are at the root level of the downloaded ZIP file. If they are not, unzip the file, and create a new ZIP file that has these files at the root level.
3. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
4. Choose **Create Bucket**.
5. On the **Create a Bucket - Select a Bucket Name and Region** page, for **Bucket Name**, type a unique name for your bucket. Bucket names must be unique across all AWS accounts, not just in your own account. This walkthrough uses the name **`my-appbucket`**, but you can use `my-appbucket-`*`yearmonthday`* to make your bucket name unique. From the **Region** drop-down list, choose **US Standard**, and then choose **Create**. **US Standard** is equivalent to `us-east-1`.



6. Choose the bucket that you created from the **All Buckets** list.
7. On the bucket page, choose **Upload**.
8. On the **Upload - Select Files and Folders** page, choose **Add files**. Browse for the ZIP file you saved in step 1, choose **Open**, and then choose **Start Upload**.



9. After the upload is complete, select the ZIP file from the list of files in your bucket, and then choose **Properties**.
10. In the **Properties** pane, copy the link to your ZIP file, and make a note of the link. You will need the bucket name and the ZIP file name portion of this link to create your pipeline.

Step 3: Add your app to OpsWorks Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Before you create a pipeline in CodePipeline, add the PHP test app to OpsWorks Stacks. When you create the pipeline, you will need to select the app that you've added to OpsWorks Stacks.

Have the Amazon S3 bucket link from step 10 of the preceding procedure ready. You will need the link to the bucket in which you stored your test app to complete this procedure.

To add an app to OpsWorks Stacks

1. In the OpsWorks Stacks console, open **MyStack**, and in the navigation pane, choose **Apps**.
2. Choose **Add app**.
3. On the **Add App** page, provide the following information:

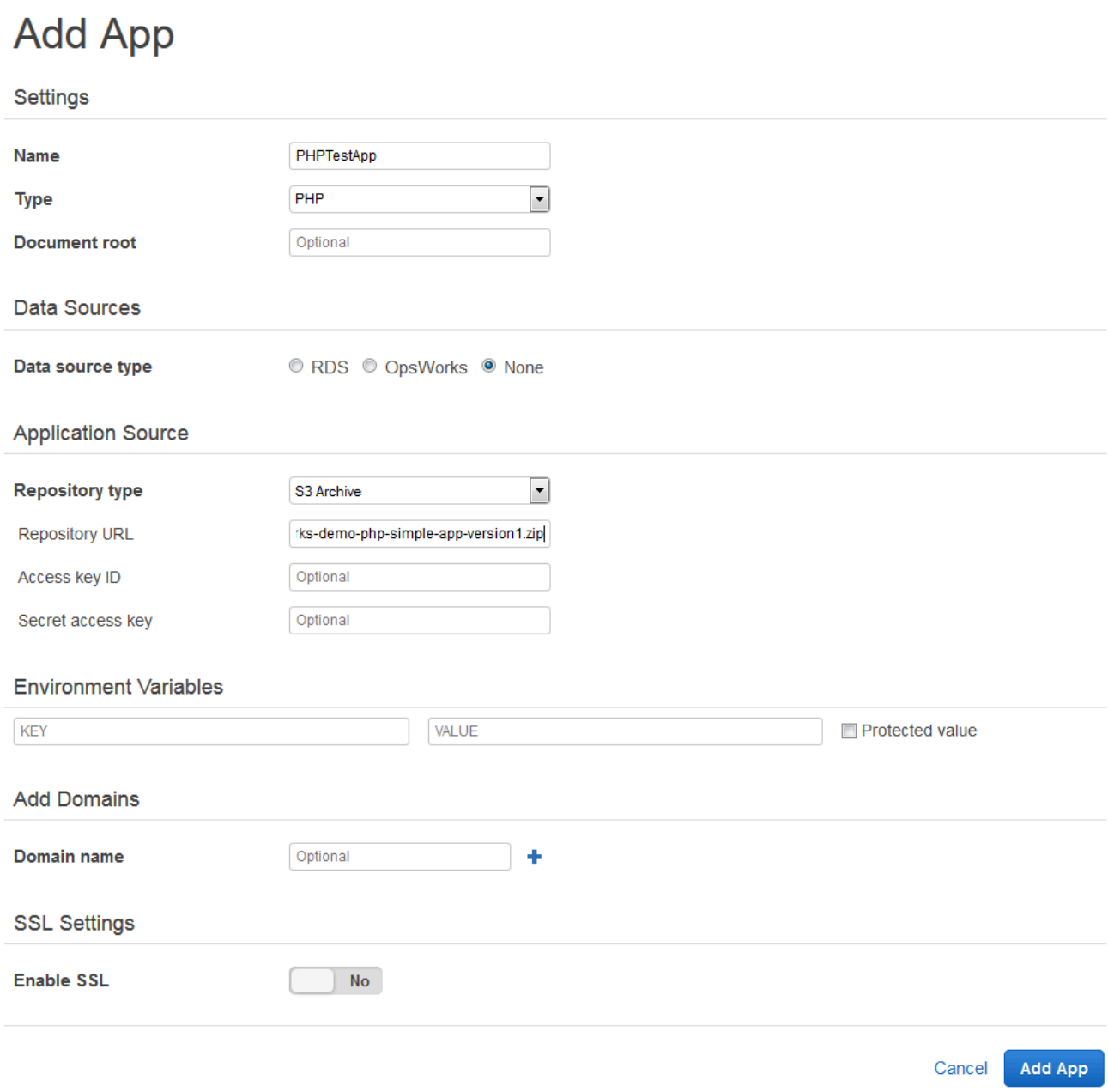
   a. Specify a name for your app. This walkthrough uses the name `PHPTestApp`.

   b. In the **Type** drop-down list, choose **PHP**.

   c. For **Data source type**, choose **None**. This app does not require an external database or data source.

   d. In the **Repository type** drop-down list, choose **S3 Archive**.

   e. In the **Repository URL** string box, paste the URL that you copied in step 10 of Step 2: Upload app code to an Amazon S3 bucket. Your form should be similar to the following:



4. You do not need to change any other settings in this form. Choose **Add App**.
5. When the **PHPTestApp** app appears in the list on the **Apps** page, continue to the next procedure, Step 4: Create a pipeline in CodePipeline.

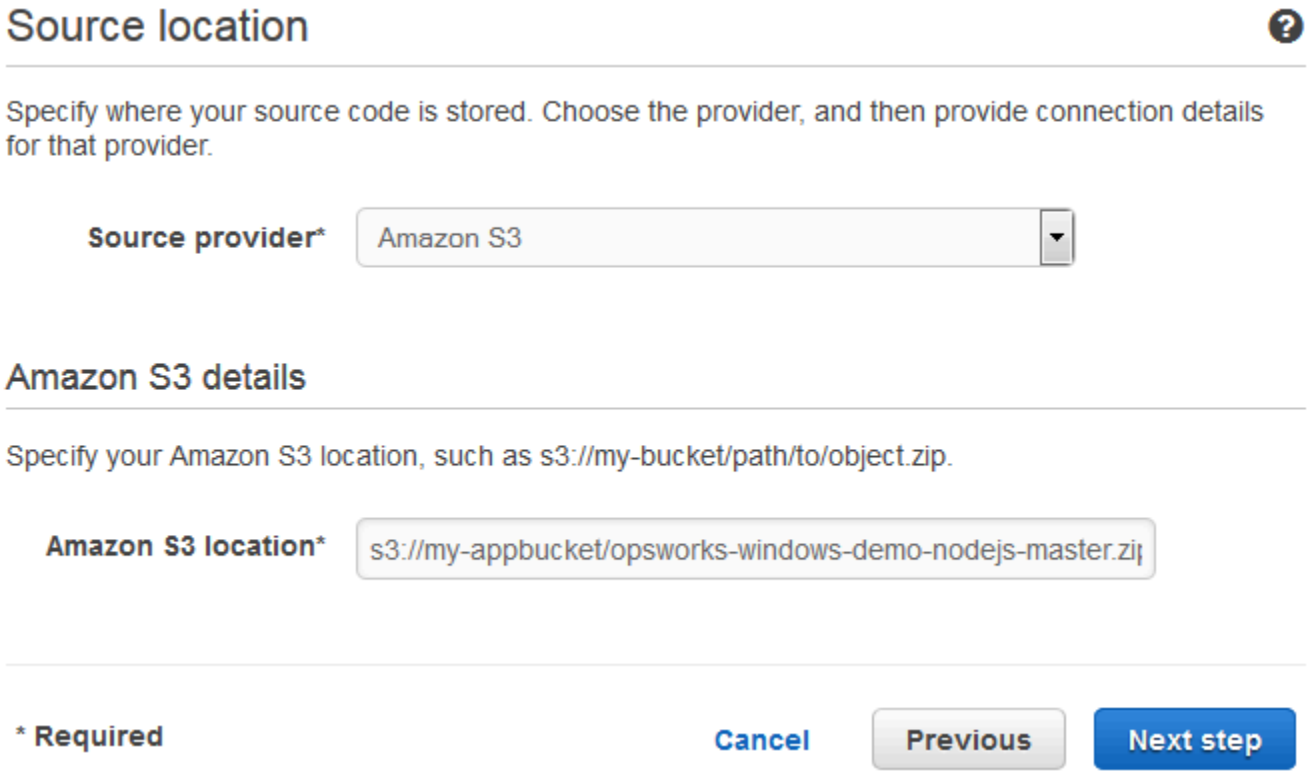
Step 4: Create a pipeline in CodePipeline

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

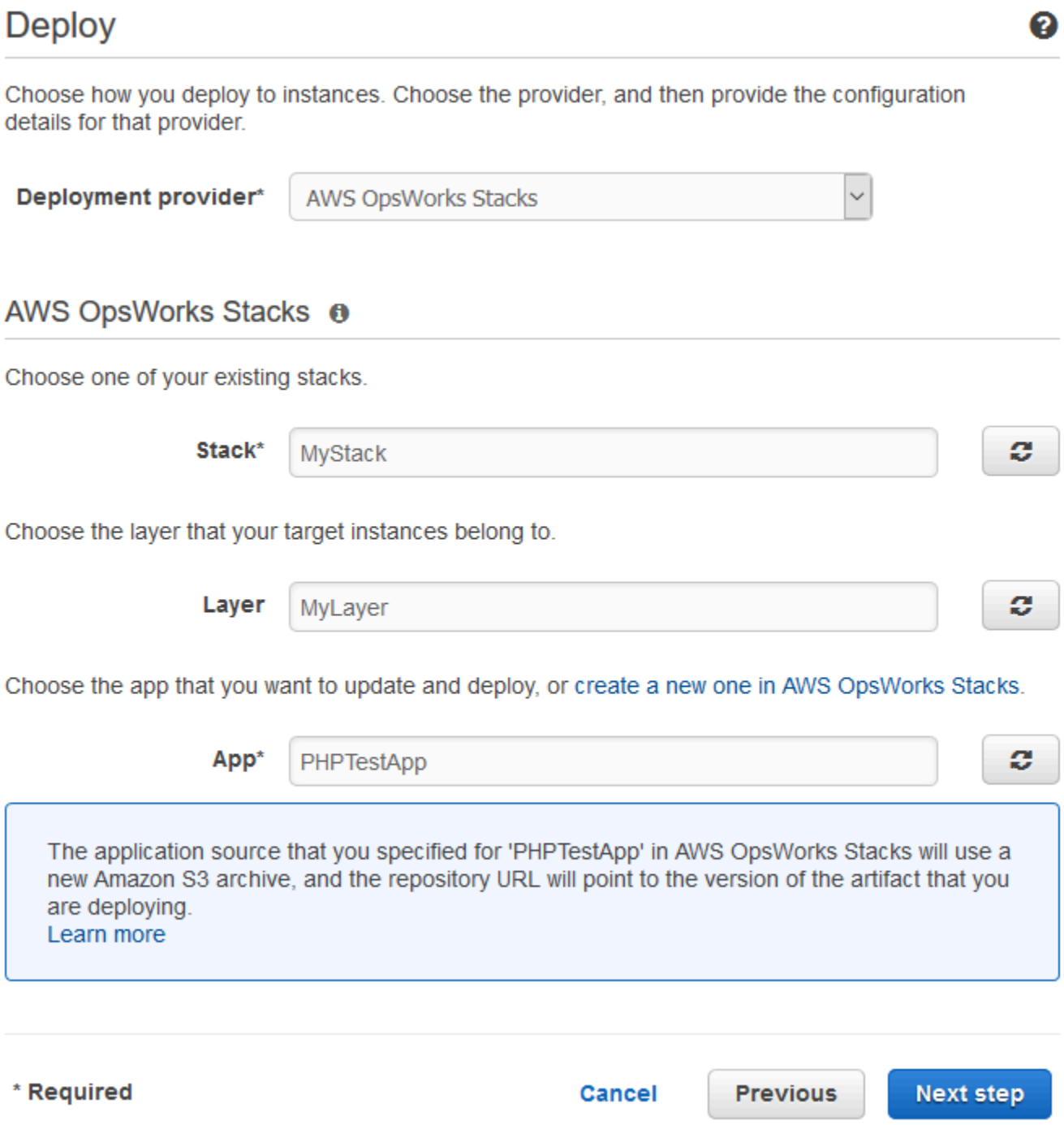
After you have a stack with a layer and at least one instance configured in OpsWorks Stacks, create a pipeline in CodePipeline with OpsWorks Stacks as the provider to deploy apps or Chef cookbooks to your OpsWorks Stacks resources.

To create a pipeline

1. Open the CodePipeline console at https://console.aws.amazon.com/codepipeline/.
2. Choose **Create pipeline**.
3. On the **Getting started with CodePipeline** page, type **`MyOpsWorksPipeline`**, or any other pipeline name that is unique to your account, and then choose **Next step**.
4. On the **Source Location** page, select **Amazon S3** from the **Source provider** drop-down list.
5. In the **Amazon S3 details** area, type your Amazon S3 bucket path, in the format **`s3://bucket-name/file name`**. Refer to the link that you noted in step 10 of Step 2: Upload app code to an Amazon S3 bucket. In this walkthrough, the path is `s3://my-appbucket/opsworks-demo-php-simple-app-version1.zip`. Choose **Next step**.



6. On the **Build** page, choose **No Build** from the drop-down list, and then choose **Next step**.
7. On the **Deploy** page, choose **OpsWorks Stacks** as the deployment provider.



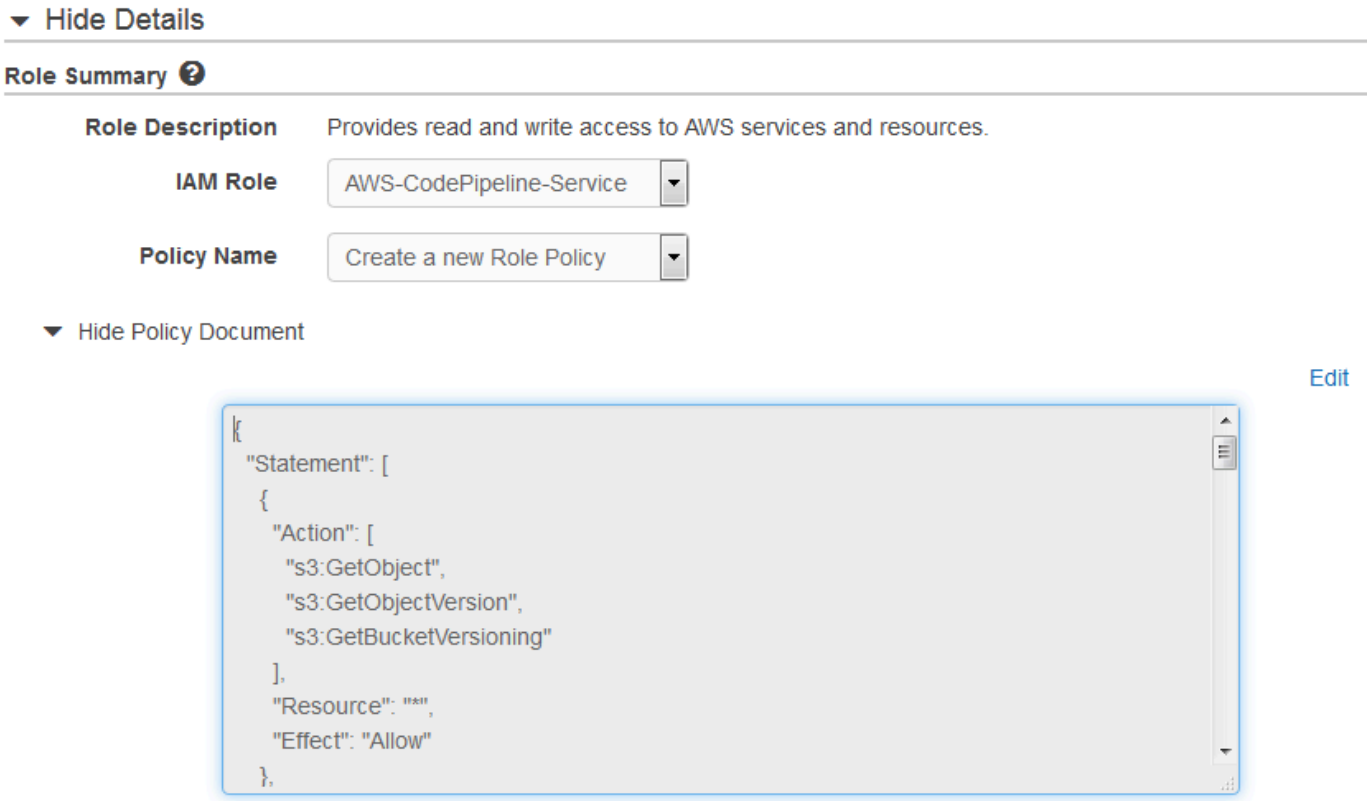
8. In the **Stack** field, type `MyStack`, or the name of the stack that you created in Step 1: Create a stack, layer, and an instance in OpsWorks Stacks.
9. In the **Layer** field, type `MyLayer`, or the name of the layer that you created in Step 1: Create a stack, layer, and an instance in OpsWorks Stacks.

10. In the **App** field, select the app that you uploaded to Amazon S3 in Step 2: Upload app code to an Amazon S3 bucket, and then choose **Next step**.
11. On the **AWS Service Role** page, choose **Create Role**.

    A new window opens with an IAM console page that describes the role that will be created for you, `AWS-CodePipeline-Service`. From the **Policy name** drop-down list, choose **Create new policy**. Be sure the policy document has the following content. Choose **Edit** to change the policy document, if required.

```
{
    "Statement": [
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:GetBucketVersioning"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": "opsworks:*",
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

    When you are finished making changes to the policy document, choose **Allow**. Your changes will be dislayed in the IAM console.



> **Note**
>
> If role creation fails, it might be because you already have an IAM role named **AWS-CodePipeline-Service**. If you have been using the **AWS-CodePipeline-Service** role before May 2016, the role might not have permissions to use OpsWorks Stacks as a deployment provider; in this case, you must update the policy statement as shown in this step. If you see an error message, go back to the beginning of this step, and choose **Use existing role** instead of **Create role**. If you use an existing role, the role should have a policy attached that includes the permissions shown in this step. For more information about the service role and its policy statement, see Edit a Policy for an IAM Service Role.

12. If the role creation process is successful, the IAM page will close, and you will be returned to the **AWS Service Role** page. Choose **Next step**.
13. On the **Review your pipeline** page, verify the choices shown on the page, and then choose **Create pipeline**.

We will create your pipeline with the following resources.

Source Stage

Source provider Amazon S3

Amazon S3 location s3://my-appbucket0/opsworks-demo-php-simple-app-version1.zip

Build Stage

Build provider No Build

Beta Stage

Deployment provider AWS OpsWorks

Stack MyStack

App PHPTestApp

Layer MyLayer

Pipeline settings

Pipeline name MyOpsWorksPipeline

Artifact location s3://codepipeline-us-east-
AWS CodePipeline will use this existing S3 bucket to store artifacts for this pipeline. Depending on the size of your artifacts, you might be charged for storage costs. For more information, see Amazon S3 storage pricing.

Role name AWS-CodePipeline-Service

To save this configuration with these resources, choose Create pipeline.

Would you like to create this pipeline?

Cancel Previous Create pipeline

14. When your pipeline is ready, it should start locating your source code and deploying your app to your stack automatically. This process can take several minutes.

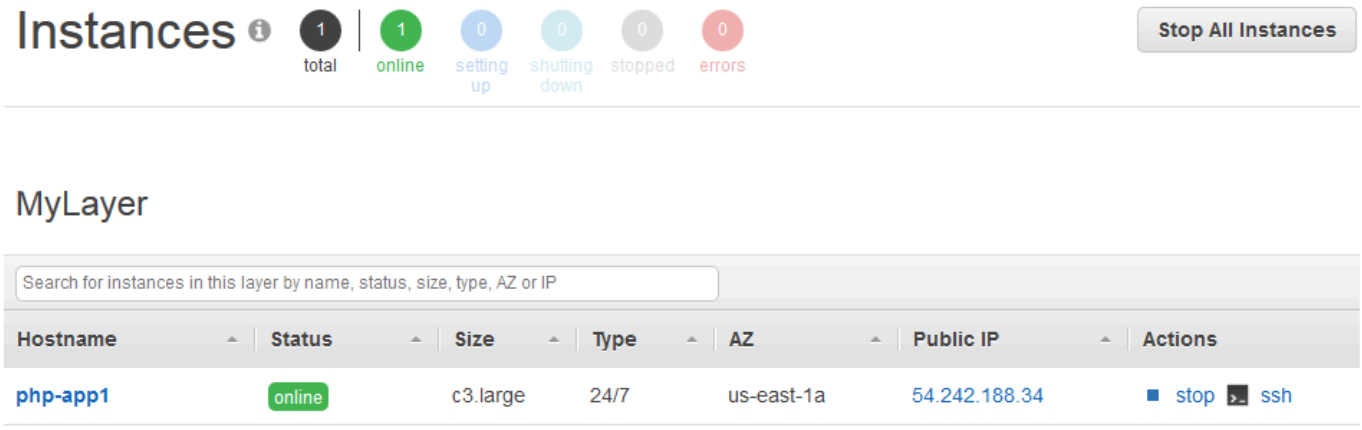
Step 5: Verifying the app deployment in OpsWorks Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

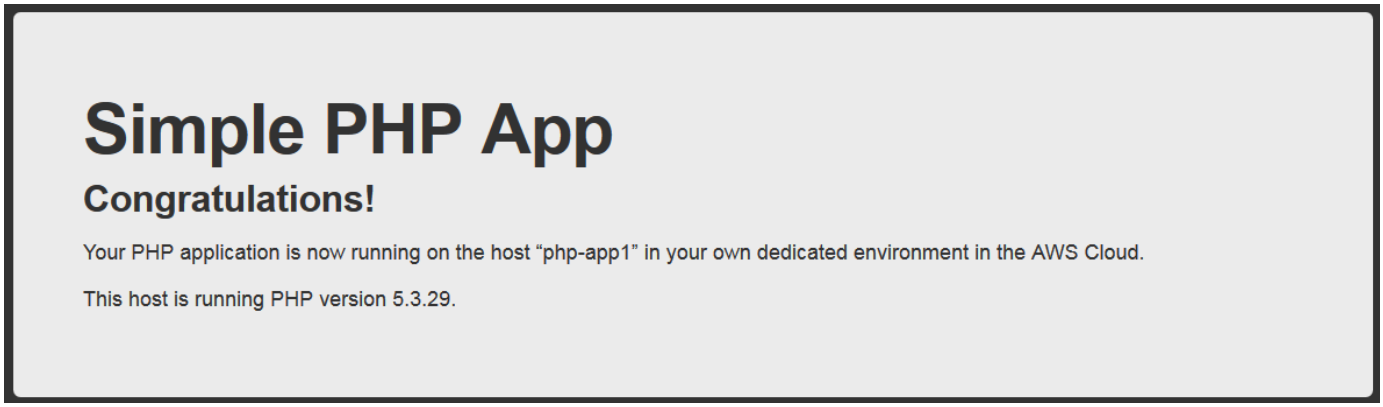
To verify that CodePipeline deployed the PHP app to your stack, sign in to the instance you created in Step 1: Create a stack, layer, and an instance in OpsWorks Stacks. You should be able to see and use the PHP web app.

To verify the app deployment in your OpsWorks Stacks instance

1. Open the OpsWorks console at https://console.aws.amazon.com/opsworks/.
2. On the OpsWorks Stacks dashboard, choose **MyStack**, and then choose **MyLayer**.
3. In the navigation pane, choose **Instances**, and then choose the public IP address of the instance that you created to view the web app.



The app will be displayed in a new browser tab.



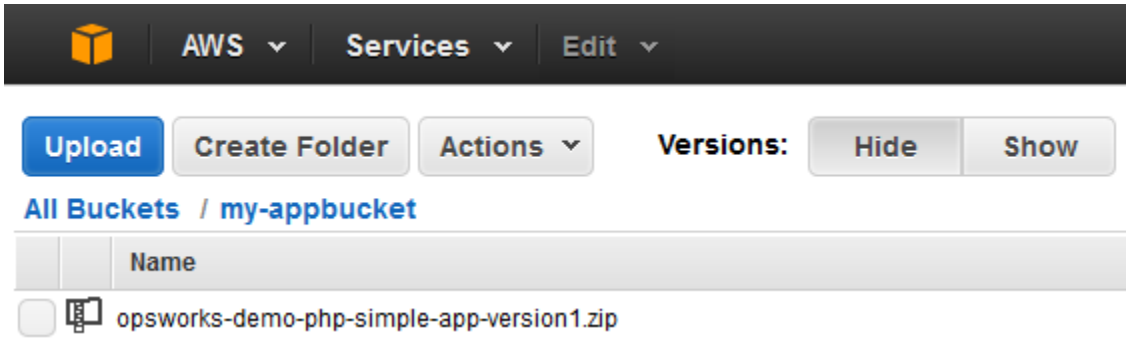
Step 6 (Optional): Update the app code to see CodePipeline redeploy your app automatically

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

When you make changes to code in apps or cookbooks that you have deployed by using CodePipeline, the updated artifacts will be deployed automatically by CodePipeline to your target instances (in this case, to a target OpsWorks Stacks stack). This section shows you the automatic redeployment when you update the code in your sample PHP app.

To edit the code in the sample app

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Open the bucket in which you are storing your sample PHP app.



3. Select the ZIP file that contains the app. On the **Actions** menu, choose **Download**.
4. In the dialog box, open the context (right-click) menu, choose **Download**, and then save the ZIP file to a convenient location. Choose **OK**.
5. Extract the contents of the ZIP file to a convenient location. You might need to change permissions on the extracted folder and its subfolders and contents to allow editing. In the `opsworks-demo-php-simple-app-version1` folder, open the `index.php` file for editing.
6. Search for the phrase, `Your PHP application is now running`. Replace the text, `Your PHP application is now running` with `You've just deployed your first app to AWS OpsWorks with AWS CodePipeline,`. Do not edit the variables.

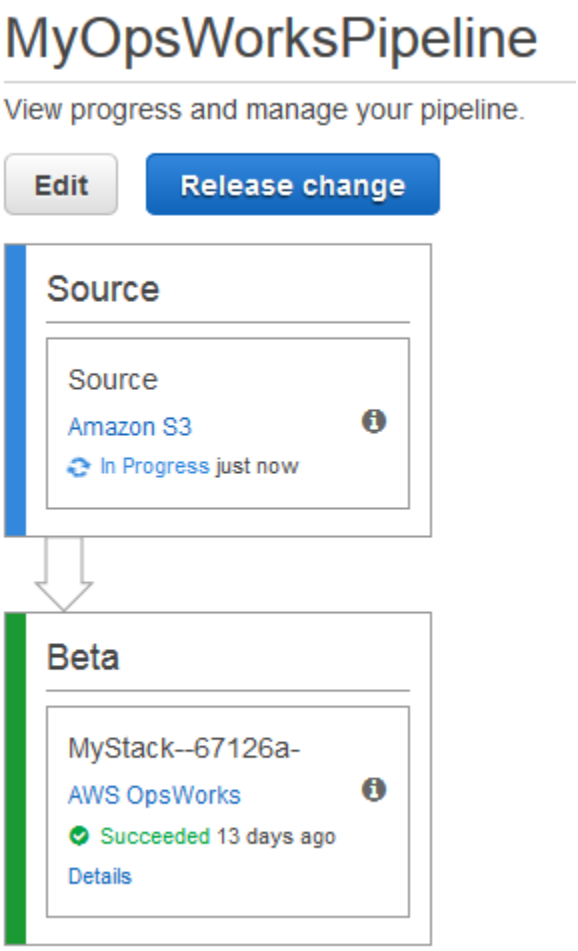
```
<body>
    <div class="container">
        <div class="hero-unit">
            <h1>Simple PHP App</h1>
            <h2>Congratulations!</h2>
            <p>You've just deployed your first app to AWS OpsWorks with AWS CodePipeline,</p>
            <p>on the host &ldquo;<?php echo gethostname(); ?>&rdquo; </p>
            <p>in your own dedicated environment in the AWS Cloud.</p>
            <p>This host is running PHP version <?php echo phpversion(); ?>.</p>
        </div>
    </div>

    <script src="//ajax.googleapis.com/ajax/libs/jquery/1.8.3/jquery.min.js"></script>
    <script src="assets/js/bootstrap.min.js"></script>
</body>
```

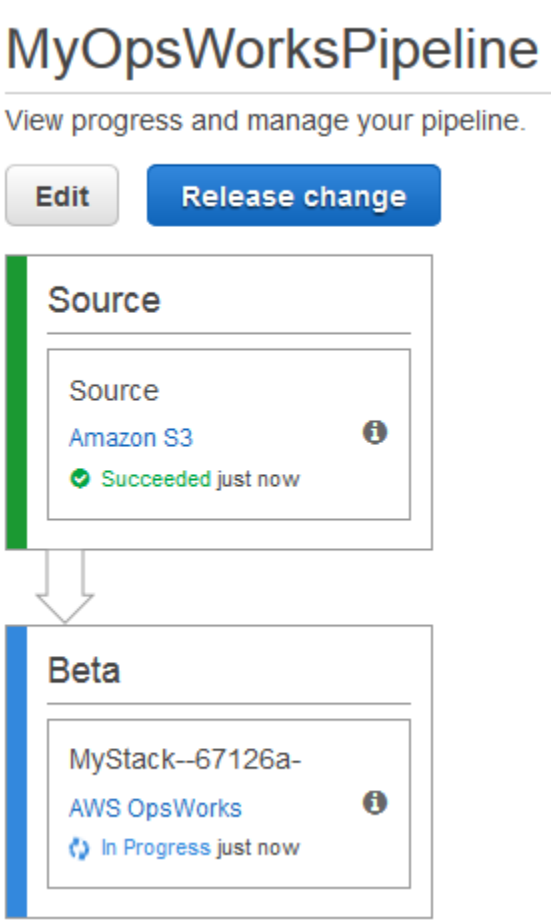
7. Save and close the `index.php` file.
8. Zip the `opsworks-demo-php-simple-app-version1` folder, and save the ZIP file to a convenient location. Do not change the name of the ZIP file.
9. Upload the new ZIP file to your Amazon S3 bucket. In this walkthrough, the name of the bucket is `my-appbucket`.
10. Open the CodePipeline console, and open your OpsWorks Stacks pipeline (**MyOpsWorksPipeline**). Choose **Release Change**.

(You can wait for CodePipeline to detect the code change from the updated version of the app in your Amazon S3 bucket. To save you time, this walkthrough instructs you to simply choose **Release Change**.)

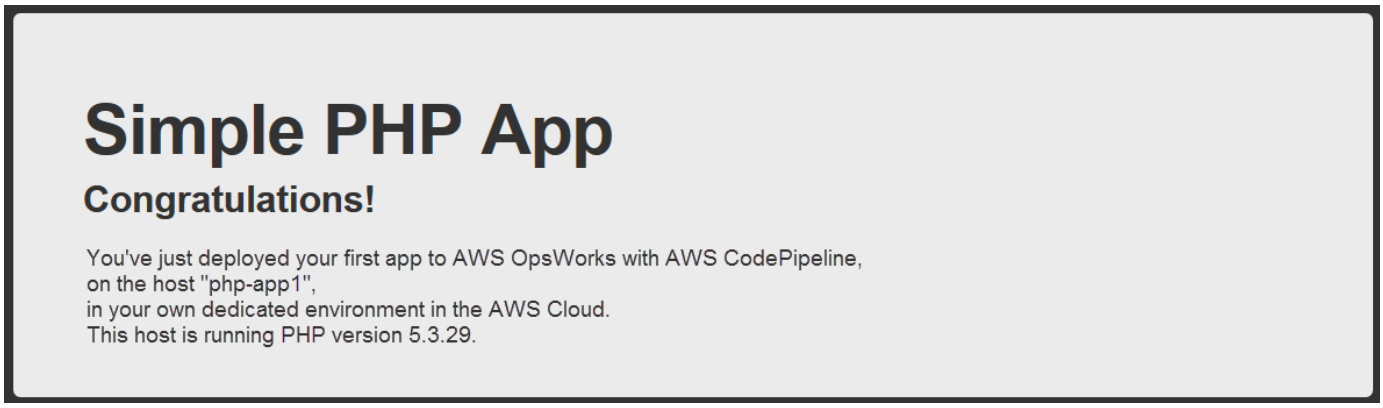
11. Observe as CodePipeline runs through the stages of the pipeline. First, CodePipeline detects changes to the source artifact.



CodePipeline pushes the updated code to your stack in OpsWorks Stacks.



12. When both stages of the pipeline have been successfully completed, open your stack in OpsWorks Stacks (**MyStack**).
13. On the **MyStack** properties page, choose **Instances**.
14. In the **Public IP** column, choose the public IP address of your instance to view the updated app's text.



Step 7 (Optional): Clean up resources

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

To help prevent unwanted charges to your AWS account, you can delete the AWS resources that you used for this walkthrough. These AWS resources include the OpsWorks Stacks stack, the IAM role and instance profile, and the pipeline that you created in CodePipeline. However, you might want to keep using these AWS resources as you continue to learn more about OpsWorks Stacks and CodePipeline. If you want to keep these resources, you have finished this walkthrough.

To delete the app from the stack

Because you did not create or apply the app as part of your AWS CloudFormation template, delete the PHP test app before you delete the stack in AWS CloudFormation.

1. In the OpsWorks Stacks console, in the service navigation pane, choose **Apps**.
2. On the **Apps** page, select **PHPTestApp**, and then in **Actions**, choose **delete**. When you are prompted to confirm, choose **Delete**. OpsWorks Stacks will delete the app.

To delete the stack

Because you created the stack by running an AWS CloudFormation template, you can delete the stack, including the layer, instance, instance profile, and security group that the template created, in the AWS CloudFormation console.

1. Open the AWS CloudFormation console.
2. In the AWS CloudFormation console dashboard, select the stack you created (**MyStack**). On the **Actions** menu, choose **Delete Stack**. When you are prompted to confirm, choose **Yes, Delete**.
3. Wait for **DELETE_COMPLETE** to appear in the **Status** column for the stack.

To delete the pipeline

1. Open the CodePipeline console.
2. In the CodePipeline dashboard, choose the pipeline you created for this walkthrough.
3. On the pipeline page, choose **Edit**.
4. On the **Edit** page, choose **Delete**. When you are prompted to confirm, choose **Delete**.

Using the OpsWorks Stacks CLI

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The OpsWorks Stacks command line interface (CLI) provides the same functionality as the console and can be used for a variety of tasks. The OpsWorks Stacks CLI is part of the AWS CLI. For more information, including how to install and configure the AWS CLI, go to What Is the AWS Command Line Interface?. For a complete description of each command, go to the OpsWorks Stacks reference.

> **Note**
>
> If you are using a Windows-based workstation, you can also run the AWS Tools for Windows PowerShell to perform OpsWorks Stacks operations from the command line. For more information, see AWS Tools for Windows PowerShell.

OpsWorks Stacks commands have the following general format:

```
aws opsworks --region us-west-1 opsworks command-name [--argument1 value] [...]
```

If an argument value is a JSON object, you should escape the " characters or the command might return an error that the JSON is not valid. For example, if the JSON object is

`"{"somekey":"somevalue"}"`, you should format it as `"{\"somekey\":\"somevalue\"}"`. An alternative approach is to put the JSON object in a file and use `file://` to include it in the command line. The following example creates an app using an application source object stored in appsource.json.

```
aws opsworks --region us-west-1 create-app --stack-id 8c428b08-a1a1-46ce-a5f8-
feddc43771b8 --name SimpleJSP --type java --app-source file://appsource.json
```

Most commands return one or more values, packaged as a JSON object. The following sections contain some examples. For a detailed description of the return values for each command, go to the OpsWorks Stacks reference.

Note

AWS CLI commands must specify a region, as shown in the examples. Valid values for the --region parameter are shown in the following table. To simplify your OpsWorks Stacks command strings, configure the CLI to specify your default region, so you can omit the `--region` parameter. If you typically work in multiple regional endpoints, do not configure the AWS CLI to use a default regional endpoint. The Canada (Central) Region endpoint is available in the API and AWS CLI only; it is not available for stacks that you create in the AWS Management Console. For more information, see Configuring the AWS Region.

Region name	Command code
US East (Ohio) Region	us-east-2
US East (N. Virginia) Region	us-east-1
US West (N. California) Region	us-west-1
US West (Oregon) Region	us-west-2
Canada (Central) Region	ca-central-1
Europe (Ireland) Region	eu-west-1
Europe (London) Region	eu-west-2
Europe (Paris) Region	eu-west-3

Region name	Command code
Europe (Frankfurt) Region	eu-central-1
Asia Pacific (Tokyo) Region	ap-northeast-1
Asia Pacific (Seoul) Region	ap-northeast-2
Asia Pacific (Mumbai) Region	ap-south-1
Asia Pacific (Singapore) Region	ap-southeast-1
Asia Pacific (Sydney) Region	ap-southeast-2
South America (São Paulo) Region	sa-east-1

To use a CLI command, you must have the appropriate permissions. For more information on OpsWorks Stacks permissions, see Managing User Permissions. To determine the permissions required for a particular command, see the command's reference page in the OpsWorks Stacks reference.

The following sections describe how to use the OpsWorks Stacks CLI to perform a variety of common tasks.

Create an Instance (create-instance)

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Use the create-instance command to create an instance on a specified stack.

Topics

- Create an Instance with a Default Host Name
- Create an Instance with a Themed Host Name
- Create an Instance with a Custom AMI

Create an Instance with a Default Host Name

```
C:\>aws opsworks --region us-west-1 create-instance --stack-id 935450cc-61e0-4b03-
a3e0-160ac817d2bb
    --layer-ids 5c8c272a-f2d5-42e3-8245-5bf3927cb65b --instance-type m1.large --os
 "Amazon Linux"
```

The arguments are as follows:

- `stack-id` – You can get the stack ID from the stack's settings page on the console (look for **OpsWorks ID**) or by calling describe-stacks.
- `layer-ids` – You can get layer IDs from the layer's details page on the console (look for **OpsWorks ID**) or by calling describe-layers. In this example, the instance belongs to only one layer.
- `instance-type` – The specification that defines the memory, CPU, storage capacity, and hourly cost for the instance; `m1.large` for this example.
- `os` – The instance's operating system; Amazon Linux for this example.

The command returns a JSON object that contains the instance ID, as follows:

```
{
    "InstanceId": "5f9adeaa-c94c-42c6-aeef-28a5376002cd"
}
```

This example creates an instance with a default host name, which is simply an integer. The following section describes how to create an instance with a host name generated from a theme.

Create an Instance with a Themed Host Name

You can also create an instance with a themed host name. You specify the theme when you create the stack. For more information, see Create a New Stack.To create the instance, first call get-hostname-suggestion to generate a name. For example:

```
C:\>aws opsworks get-hostname-suggestion --region us-west-1 --layer-id 5c8c272a-
f2d5-42e3-8245-5bf3927cb65b
```

If you specify the default `Layer Dependent` theme, `get-hostname-suggestion` simply appends a digit to the layer's short name. For more information, see Create a New Stack.

The command returns the generated host name.

```
{
    "Hostname": "php-app2",
    "LayerId": "5c8c272a-f2d5-42e3-8245-5bf3927cb65b"
}
```

You can then use the `hostname` argument to pass the generated name to `create-instance`, as follows:

```
c:\>aws --region us-west-1 opsworks create-instance --stack-id 935450cc-61e0-4b03-
a3e0-160ac817d2bb
   --layer-ids 5c8c272a-f2d5-42e3-8245-5bf3927cb65b --instance-type m1.large --os
 "Amazon Linux" --hostname "php-app2"
```

Create an Instance with a Custom AMI

The following create-instance command creates an instance with a custom AMI, which must be from the stack's region. For more information about how to create a custom AMI for OpsWorks Stacks, see Using Custom AMIs.

```
C:\>aws opsworks create-instance --region us-west-1 --stack-id c5ef46ce-3ccd-472c-
a3de-9bec94c6028e
   --layer-ids 6ff8a2ac-c9cc-49cf-9c67-fc852539ade4 --instance-type c3.large --os
 Custom
```

```
    --ami-id ami-6c61f104
```

The arguments are as follows:

- `stack-id` – You can get the stack ID from the stack's settings page on the console (look for **OpsWorks ID**) or by calling describe-stacks.
- `layer-ids` – You can get layer IDs from the layer's details page on the console (look for **OpsWorks ID**) or by calling describe-layers. In this example, the instance belongs to only one layer.
- `instance-type` – The value defines the instance's memory, CPU, storage capacity, and hourly cost, and must be compatible with the AMI (`c3.large` for this example).
- `os` – The instance's operating system, which must be set to `Custom` for a custom AMI.
- `ami-id` – The AMI ID, which should look something like `ami-6c61f104`

> **Note**
>
> When you use a custom AMI, block device mappings are not supported, and values that you specify for the `--block-device-mappings` option are ignored.

The command returns a JSON object that contains the instance ID, as follows:

```
{
    "InstanceId": "5f9adeaa-c94c-42c6-aeef-28a5376002cd"
}
```

Deploy an App (create-deployment)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Use the create-deployment command to deploy an app to a specified stack.

Topics

- Deploy an App

Deploy an App

```
aws opsworks --region us-west-1 create-deployment --stack-id cfb7e082-ad1d-4599-8e81-
de1c39ab45bf
   --app-id 307be5c8-d55d-47b5-bd6e-7bd417c6c7eb --command "{\"Name\":\"deploy\"}"
```

The arguments are as follows:

- `stack-id` – You can get the stack ID from the stack's settings page on the console (look for **OpsWorks ID**) or by calling `describe-stacks`.
- `app-id` – You can get app ID from the app's details page (look for **OpsWorks ID**) or by calling describe-apps.
- `command` – The argument takes a JSON object that sets the command name to `deploy`, which deploys the specified app to the stack.

  Notice that the " characters in the JSON object are all escaped. Otherwise, the command might return an error that the JSON is invalid.

The command returns a JSON object that contains the deployment ID, as follows:

```
{
    "DeploymentId": "5746c781-df7f-4c87-84a7-65a119880560"
}
```

> **Note**
>
> The preceding example deploys to every instance in the stack. To deploy to a specified subset of instances, add an `instance-ids` argument and list the instance IDs.

List a Stack's Apps (describe-apps)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Use the describe-apps command to list a stack's apps or get details about specified apps.

```
aws opsworks --region us-west-1 describe-apps --stack-id 38ee91e2-abdc-4208-
a107-0b7168b3cc7a
```

The preceding example returns a JSON object that contains information about each app. This example has only one app. For a description of each parameter, see describe-apps.

```
{
  "Apps": [
    {
      "StackId": "38ee91e2-abdc-4208-a107-0b7168b3cc7a",
      "AppSource": {
        "Url": "url",
        "Type": "archive"
      },
      "Name": "SimpleJSP",
      "EnableSsl": false,
      "SslConfiguration": {},
      "AppId": "da1decc1-0dff-43ea-ad7c-bb667cd87c8b",
      "Attributes": {
        "RailsEnv": null,
        "AutoBundleOnDeploy": "true",
        "DocumentRoot": "ROOT"
      },
      "Shortname": "simplejsp",
      "Type": "other",
      "CreatedAt": "2013-08-01T21:46:54+00:00"
    }
```

```
  ]
}
```

List a Stack's Commands (describe-commands)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Use the describe-commands command to list a stack's commands or get details about specified commands. The following example gets information about the commands that have been executed on a specified instance.

```
aws opsworks --region us-west-1 describe-commands --instance-id
 8c2673b9-3fe5-420d-9cfa-78d875ee7687
```

The command returns a JSON object that contains details about each command. The Type parameter identifies the command name, deploy or undeploy for this example. For a description of the other parameters, see describe-commands.

```
{
  "Commands": [
    {
      "Status": "successful",
      "CompletedAt": "2013-07-25T18:57:47+00:00",
      "InstanceId": "8c2673b9-3fe5-420d-9cfa-78d875ee7687",
      "DeploymentId": "6ed0df4c-9ef7-4812-8dac-d54a05be1029",
      "AcknowledgedAt": "2013-07-25T18:57:41+00:00",
      "LogUrl": "https://s3.amazonaws.com/prod_stage-log/logs/008c1a91-
ec59-4d51-971d-3adff54b00cc?AWSAccessKeyId=AIDACKCEVSQ6C2EXAMPLE
 &Expires=1375394373&Signature=HkXil6UuNfxTCC37EPQAa462E1E%3D&response-cache-
control=private&response-content-encoding=gzip&response-content- type=text%2Fplain",
      "Type": "undeploy",
      "CommandId": "008c1a91-ec59-4d51-971d-3adff54b00cc",
```

```
      "CreatedAt": "2013-07-25T18:57:34+00:00",
      "ExitCode": 0
    },
    {
      "Status": "successful",
      "CompletedAt": "2013-07-25T18:55:40+00:00",
      "InstanceId": "8c2673b9-3fe5-420d-9cfa-78d875ee7687",
      "DeploymentId": "19d3121e-d949-4ff2-9f9d-94eac087862a",
      "AcknowledgedAt": "2013-07-25T18:55:32+00:00",
      "LogUrl": "https://s3.amazonaws.com/prod_stage-log/
logs/899d3d64-0384-47b6-a586-33433aad117c?AWSAccessKeyId=AIDACKCEVSQ6C2EXAMPLE
 &Expires=1375394373&Signature=xMsJvtLuUqWmsr8s%2FAjVru0BtRs%3D&response-cache-
control=private&response-content-encoding=gzip&response-conten t-type=text%2Fplain",
      "Type": "deploy",
      "CommandId": "899d3d64-0384-47b6-a586-33433aad117c",
      "CreatedAt": "2013-07-25T18:55:29+00:00",
      "ExitCode": 0
    }
  ]
}
```


List a Stack's Deployments (describe-deployments)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Use the describe-deployments command to list a stack's deployments or get details about specified deployments.

```
aws opsworks --region us-west-1 describe-deployments --stack-id 38ee91e2-abdc-4208-
a107-0b7168b3cc7a
```

The preceding command returns a JSON object that contains details about each deployment for the specified stack. For a description of each parameter, see describe-deployments.

```
{
  "Deployments": [
      {
          "StackId": "38ee91e2-abdc-4208-a107-0b7168b3cc7a",
          "Status": "successful",
          "CompletedAt": "2013-07-25T18:57:49+00:00",
          "DeploymentId": "6ed0df4c-9ef7-4812-8dac-d54a05be1029",
          "Command": {
              "Args": {},
              "Name": "undeploy"
          },
          "CreatedAt": "2013-07-25T18:57:34+00:00",
          "Duration": 15,
          "InstanceIds": [
              "8c2673b9-3fe5-420d-9cfa-78d875ee7687",
              "9e588a25-35b2-4804-bd43-488f85ebe5b7"
          ]
      },
      {
          "StackId": "38ee91e2-abdc-4208-a107-0b7168b3cc7a",
          "Status": "successful",
          "CompletedAt": "2013-07-25T18:56:41+00:00",
          "IamUserArn": "arn:aws:iam::444455556666:user/example-user",
          "DeploymentId": "19d3121e-d949-4ff2-9f9d-94eac087862a",
          "Command": {
              "Args": {},
              "Name": "deploy"
          },
          "InstanceIds": [
              "8c2673b9-3fe5-420d-9cfa-78d875ee7687",
              "9e588a25-35b2-4804-bd43-488f85ebe5b7"
          ],
          "Duration": 72,
          "CreatedAt": "2013-07-25T18:55:29+00:00"
      }
  ]
}
```

List a Stack's Elastic IP Addresses (describe-elastic-ips)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Use the describe-elastic-ips command to list the Elastic IP addresses that have been registered with a stack or get details about specified Elastic IP addresses.

```
aws opsworks --region us-west-2 describe-elastic-ips --instance-id b62f3e04-e9eb-436c-a91f-d9e9a396b7b0
```

The preceding command returns a JSON object that contains details about each Elastic IP address (one in this example) for a specified instance. For a description of each parameter, see describe-elastic-ips.

```
{
  "ElasticIps": [
      {
          "Ip": "192.0.2.0",
          "Domain": "standard",
          "Region": "us-west-2"
      }
  ]
}
```

List a Stack's Instances (describe-instances)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about

migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Use the describe-instances command to list a stack's instances or get details about specified instances.

```
C:\>aws opsworks --region us-west-2 describe-instances --stack-id 38ee91e2-abdc-4208-
a107-0b7168b3cc7a
```

The preceding command returns a JSON object that contains details about every instance in a specified stack. For a description of each parameter, see describe-instances.

```
{
  "Instances": [
      {
          "StackId": "38ee91e2-abdc-4208-a107-0b7168b3cc7a",
          "SshHostRsaKeyFingerprint":
 "f4:3b:8e:27:1b:73:98:80:5d:d7:33:e2:b8:c8:8f:de",
          "Status": "stopped",
          "AvailabilityZone": "us-west-2a",
          "SshHostDsaKeyFingerprint":
 "e8:9b:c7:02:18:2a:bd:ab:45:89:21:4e:af:0b:07:ac",
          "InstanceId": "8c2673b9-3fe5-420d-9cfa-78d875ee7687",
          "Os": "Amazon Linux",
          "Hostname": "db-master1",
          "SecurityGroupIds": [],
          "Architecture": "x86_64",
          "RootDeviceType": "instance-store",
          "LayerIds": [
              "41a20847-d594-4325-8447-171821916b73"
          ],
          "InstanceType": "c1.medium",
          "CreatedAt": "2013-07-25T18:11:27+00:00"
      },
      {
          "StackId": "38ee91e2-abdc-4208-a107-0b7168b3cc7a",
          "SshHostRsaKeyFingerprint":
 "ae:3a:85:54:66:f3:ce:98:d9:83:39:1e:10:a9:38:12",
          "Status": "stopped",
          "AvailabilityZone": "us-west-2a",
```

```
          "SshHostDsaKeyFingerprint":
 "5b:b9:6f:5b:1c:ec:55:85:f3:45:f1:28:25:1f:de:e4",
          "InstanceId": "9e588a25-35b2-4804-bd43-488f85ebe5b7",
          "Os": "Amazon Linux",
          "Hostname": "tomcustom1",
          "SecurityGroupIds": [],
          "Architecture": "x86_64",
          "RootDeviceType": "instance-store",
          "LayerIds": [
              "e6cbcd29-d223-40fc-8243-2eb213377440"
          ],
          "InstanceType": "c1.medium",
          "CreatedAt": "2013-07-25T18:15:52+00:00"
      }
  ]
}
```

List an Account's Stacks (describe-stacks)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Use the describe-stacks command to list an account's stacks or get details about specified stacks.

```
aws opsworks --region us-west-2 describe-stacks
```

The preceding command returns a JSON object that contains details about each stack in the account, two in this example. For a description of each parameter, see describe-stacks.

```
{
    "Stacks": [
        {
            "ServiceRoleArn": "arn:aws:iam::444455556666:role/aws-opsworks-service-
role",
            "StackId": "aeb7523e-7c8b-49d4-b866-03aae9d4fbcb",
```

```
            "DefaultRootDeviceType": "instance-store",
            "Name": "TomStack-sd",
            "ConfigurationManager": {
                "Version": "11.4",
                "Name": "Chef"
            },
            "UseCustomCookbooks": true,
            "CustomJson": "{\n  \"tomcat\": {\n    \"base_version\": 7,\n
 \"java_opts\": \"-Djava.awt.headless=true -Xmx256m\"\n  },\n  \
"datasources\": {\n    \"ROOT\": \"jdbc/mydb\"\n  }\n}",
            "Region": "us-west-2",
            "DefaultInstanceProfileArn": "arn:aws:iam::444455556666:instance-profile/
aws-opsworks-ec2-role",
            "CustomCookbooksSource": {
                "Url": "git://github.com/example-repo/tomcustom.git",
                "Type": "git"
            },
            "DefaultAvailabilityZone": "us-west-2a",
            "HostnameTheme": "Layer_Dependent",
            "Attributes": {
                "Color": "rgb(45, 114, 184)"
            },
            "DefaultOs": "Amazon Linux",
            "CreatedAt": "2013-08-01T22:53:42+00:00"
        },
        {
            "ServiceRoleArn": "arn:aws:iam::444455556666:role/aws-opsworks-service-
role",
            "StackId": "40738975-da59-4c5b-9789-3e422f2cf099",
            "DefaultRootDeviceType": "instance-store",
            "Name": "MyStack",
            "ConfigurationManager": {
                "Version": "11.4",
                "Name": "Chef"
            },
            "UseCustomCookbooks": false,
            "Region": "us-west-2",
            "DefaultInstanceProfileArn": "arn:aws:iam::444455556666:instance-profile/
aws-opsworks-ec2-role",
            "CustomCookbooksSource": {},
            "DefaultAvailabilityZone": "us-west-2a",
            "HostnameTheme": "Layer_Dependent",
            "Attributes": {
                "Color": "rgb(45, 114, 184)"
```

```
            },
            "DefaultOs": "Amazon Linux",
            "CreatedAt": "2013-10-25T19:24:30+00:00"
        }
    ]
}
```

List a Stack's Layers (describe-layers)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Use the describe-layers command to list a stack's layers or get details about specified layers.

```
aws opsworks --region us-west-2 describe-layers --stack-id 38ee91e2-abdc-4208-
a107-0b7168b3cc7a
```

The preceding command returns a JSON object that contains details about each layer in a specified stack—in this example, a MySQL layer and a custom layer. For a description of each parameter, see describe-layers.

```
{
  "Layers": [
      {
          "StackId": "38ee91e2-abdc-4208-a107-0b7168b3cc7a",
          "Type": "db-master",
          "DefaultSecurityGroupNames": [
              "AWS-OpsWorks-DB-Master-Server"
          ],
          "Name": "MySQL",
          "Packages": [],
          "DefaultRecipes": {
              "Undeploy": [],
              "Setup": [
                  "opsworks_initial_setup",
```

```
                "ssh_host_keys",
                "ssh_users",
                "mysql::client",
                "dependencies",
                "ebs",
                "opsworks_ganglia::client",
                "mysql::server",
                "dependencies",
                "deploy::mysql"
            ],
            "Configure": [
                "opsworks_ganglia::configure-client",
                "ssh_users",
                "agent_version",
                "deploy::mysql"
            ],
            "Shutdown": [
                "opsworks_shutdown::default",
                "mysql::stop"
            ],
            "Deploy": [
                "deploy::default",
                "deploy::mysql"
            ]
        },
        "CustomRecipes": {
            "Undeploy": [],
            "Setup": [],
            "Configure": [],
            "Shutdown": [],
            "Deploy": []
        },
        "EnableAutoHealing": false,
        "LayerId": "41a20847-d594-4325-8447-171821916b73",
        "Attributes": {
            "MysqlRootPasswordUbiquitous": "true",
            "RubygemsVersion": null,
            "RailsStack": null,
            "HaproxyHealthCheckMethod": null,
            "RubyVersion": null,
            "BundlerVersion": null,
            "HaproxyStatsPassword": null,
            "PassengerVersion": null,
            "MemcachedMemory": null,
```

```
            "EnableHaproxyStats": null,
            "ManageBundler": null,
            "NodejsVersion": null,
            "HaproxyHealthCheckUrl": null,
            "MysqlRootPassword": "*****FILTERED*****",
            "GangliaPassword": null,
            "GangliaUser": null,
            "HaproxyStatsUrl": null,
            "GangliaUrl": null,
            "HaproxyStatsUser": null
        },
        "Shortname": "db-master",
        "AutoAssignElasticIps": false,
        "CustomSecurityGroupIds": [],
        "CreatedAt": "2013-07-25T18:11:19+00:00",
        "VolumeConfigurations": [
            {
                "MountPoint": "/vol/mysql",
                "Size": 10,
                "NumberOfDisks": 1
            }
        ]
    },
    {
        "StackId": "38ee91e2-abdc-4208-a107-0b7168b3cc7a",
        "Type": "custom",
        "DefaultSecurityGroupNames": [
            "AWS-OpsWorks-Custom-Server"
        ],
        "Name": "TomCustom",
        "Packages": [],
        "DefaultRecipes": {
            "Undeploy": [],
            "Setup": [
                "opsworks_initial_setup",
                "ssh_host_keys",
                "ssh_users",
                "mysql::client",
                "dependencies",
                "ebs",
                "opsworks_ganglia::client"
            ],
            "Configure": [
                "opsworks_ganglia::configure-client",
```

```
                "ssh_users",
                "agent_version"
            ],
            "Shutdown": [
                "opsworks_shutdown::default"
            ],
            "Deploy": [
                "deploy::default"
            ]
        },
        "CustomRecipes": {
            "Undeploy": [],
            "Setup": [
                "tomcat::setup"
            ],
            "Configure": [
                "tomcat::configure"
            ],
            "Shutdown": [],
            "Deploy": [
                "tomcat::deploy"
            ]
        },
        "EnableAutoHealing": true,
        "LayerId": "e6cbcd29-d223-40fc-8243-2eb213377440",
        "Attributes": {
            "MysqlRootPasswordUbiquitous": null,
            "RubygemsVersion": null,
            "RailsStack": null,
            "HaproxyHealthCheckMethod": null,
            "RubyVersion": null,
            "BundlerVersion": null,
            "HaproxyStatsPassword": null,
            "PassengerVersion": null,
            "MemcachedMemory": null,
            "EnableHaproxyStats": null,
            "ManageBundler": null,
            "NodejsVersion": null,
            "HaproxyHealthCheckUrl": null,
            "MysqlRootPassword": null,
            "GangliaPassword": null,
            "GangliaUser": null,
            "HaproxyStatsUrl": null,
            "GangliaUrl": null,
```

```
                "HaproxyStatsUser": null
            },
            "Shortname": "tomcustom",
            "AutoAssignElasticIps": false,
            "CustomSecurityGroupIds": [],
            "CreatedAt": "2013-07-25T18:12:53+00:00",
            "VolumeConfigurations": []
        }
    ]
}
```

Execute a Recipe (create-deployment)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Use the create-deployment command to execute stack commands and deployment commands. The following example executes a stack command to run a custom recipe on a specified stack.

```
aws opsworks --region us-west-1 create-deployment --stack-id 935450cc-61e0-4b03-
a3e0-160ac817d2bb
    --command "{\"Name\":\"execute_recipes\", \"Args\":{\"recipes\":[\"phpapp::appsetup
\"]}}"
```

The `command` argument takes a JSON object that is formatted as follows:

- `Name` - Specifies the command name. The `execute_recipes` command used for this example executes a specified recipe on the stack's instances.
- `Args` - Specifies a list of arguments and their values. This example has one argument, `recipes`, which is set to the recipe to be executed, `phpapp::appsetup`.

Notice that the " characters in the JSON object are all escaped. Otherwise, the command might return an error that the JSON is invalid.

The command returns a deployment ID, which you can use to identify the command for other CLI commands such as `describe-commands`.

```
{
    "DeploymentId": "5cbaa7b9-4e09-4e53-aa1b-314fbd106038"
}
```

Install Dependencies (create-deployment)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Use the create-deployment command to execute stack commands and deployment commands. The following example runs the `update_dependencies` stack command to update the dependencies on a stack's instances.

```
aws opsworks --region us-west-1 create-deployment --stack-id 935450cc-61e0-4b03-
a3e0-160ac817d2bb
--command "{\"Name\":\"install_dependencies\"}"
```

The `command` argument takes a JSON object with a `Name` parameter whose value specifies the command name, `install_dependencies` for this example. Notice that the " characters in the JSON object are all escaped. Otherwise, the command might return an error that the JSON is invalid.

The command returns a deployment ID, which you can use to identify the command for other CLI commands such as `describe-commands`.

```
{
    "DeploymentId": "aef5b255-8604-4928-81b3-9b0187f962ff"
}
```

Update the Stack Configuration (update-stack)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Use the update-stack command to update the configuration of a specified stack. The following example updates a stack to add custom JSON to the stack configuration attributes.

```
aws opsworks --region us-west-1 update-stack --stack-id 935450cc-61e0-4b03-
a3e0-160ac817d2bb
   --custom-json "{\"somekey\":\"somevalue\"}" --service-role-arn
 arn:aws:iam::444455556666:role/aws-opsworks-service-role
```

Notice that the " characters in the JSON object are all escaped. Otherwise, the command might return an error that the JSON is invalid.

> **Note**
>
> The example also specifies a service role for the stack. You must set `service-role-arn` to a valid service role ARN or the action will fail; there is no default value. You can specify the stack's current service role ARN, if you prefer, but you must do so explicitly.

The `update-stack` command does not return a value.

Debugging and Troubleshooting Guide

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate

their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

If you need to debug a recipe or troubleshoot a service issue, the best approach generally is to work through the following steps, in order:

1. Check Common Debugging and Troubleshooting Issues for your specific issue.
2. Search the OpsWorks Stacks Forum to see if the issue has been discussed there.

   The Forum includes many experienced users and is monitored by the OpsWorks Stacks team.
3. For issues with recipes, see Debugging Recipes.
4. Contact OpsWorks Stacks support or post your issue on the OpsWorks Stacks Forum.

The following section provides guidance for debugging recipes. The final section describes common debugging and troubleshooting issues and their solutions.

Note

Each Chef run produces a log, which provides a detailed description of the run and is a valuable troubleshooting resource. To specify the amount of detail in the log, add a `Chef::Log.level` statement to a custom recipe that specifies the desired log level. The default value is `:info`. The following example shows how to set the Chef log level to `:debug`, which provides the most detailed description of the run.

```
Chef::Log.level = :debug
```

For more information about viewing and interpreting Chef logs, see Chef Logs.

Topics

- Debugging Recipes
- Common Debugging and Troubleshooting Issues

Debugging Recipes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

When a lifecycle event occurs, or you run the Execute Recipes stack command, OpsWorks Stacks issues a command to the agent to initiate a Chef Solo run on the specified instances to execute the appropriate recipes, including your custom recipes. This section describes some ways that you can debug failed recipes.

Topics

- Logging in to a Failed Instance
- Chef Logs
- Using the OpsWorks Stacks Agent CLI

Logging in to a Failed Instance

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

If a recipe fails, the instance will end up in the `setup_failed` state instead of online. Even though the instance is not online as far as OpsWorks Stacks is concerned, the EC2 instance is running and it's often useful to log in to troubleshoot the issue. For example, you can check whether an application or custom cookbook is correctly installed. The OpsWorks Stacks built-in support for

SSH and RDP login is available only for instances in the online state. However, if you have assigned an SSH key pair to the instance, you can still log in, as follows:

- Linux instances – Use the SSH key pair's private key to log in with a third-party SSH client, such as OpenSSH or PuTTY.

  You can use an EC2 key pair or your personal SSH key pair for this purpose.
- Windows instances – Use the EC2 key pair's private key to retrieve the instance's Administrator password.

  Use that password to log in with your preferred RDP Client. For more information, see Logging in As Administrator. You cannot use a personal SSH key pair to retrieve an Administrator password.

Chef Logs

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Chef logs are one of your key troubleshooting resources, especially for debugging recipes. OpsWorks Stacks captures the Chef log for each command, and retains the logs for an instance's 30 most recent commands. Because the run is in debug mode, the log contains a detailed description of the Chef run, including the text that is sent to `stdout` and `stderror`. If a recipe fails, the log includes the Chef stack trace.

OpsWorks Stacks gives you several ways to view Chef logs. Once you have the log information, you can use it to debug failed recipes.

Note

You can also view a specified log's tail by using SSH to connect to the instance and running the agent CLI `show_log` command. For more information, see Displaying Chef Logs.

Topics

- Viewing a Chef Log with the Console
- Viewing a Chef Log with the CLI or API
- Viewing a Chef Log on an Instance
- Interpreting a Chef Log
- Common Chef Log Errors

Viewing a Chef Log with the Console

The simplest way to view a Chef log is to go to the instance's details page. The **Logs** section includes an entry for each event and Execute Recipes command. The following shows an instance's **Logs** section, with **configure** and **setup** commands, which correspond to Configure and Setup lifecycle events.



Click **show** in the appropriate command's **Log** column to view the corresponding Chef log. If an error occurs, OpsWorks Stacks automatically opens the log to the error, which is usually at the end of the file.

Viewing a Chef Log with the CLI or API

You can use the OpsWorks Stacks CLI `describe-commands` command or the `DescribeCommands` API action to view the logs, which are stored on an Amazon S3 bucket. The following shows how to use the CLI to view any of the current set of logs file for a specified instance. The procedure for using `DescribeCommands` is essentially similar.

To use the OpsWorks Stacks to view an instance's Chef logs

1. Open the instance's details page and copy its **OpsWorks ID** value.
2. Use the ID value to run the `describe-commands` CLI command, as follows:

```
aws opsworks describe-commands --instance-id 67bf0da2-29ed-4217-990c-d895d51812b9
```

The command returns a JSON object with an embedded object for each command that OpsWorks Stacks has executed on the instance, with the most recent first. The `Type` parameter contains the command type for each embedded object, a `configure` command, and a `setup` command in this example.


```
{
    "Commands": [
        {
            "Status": "successful",
            "CompletedAt": "2013-10-25T19:38:36+00:00",
            "InstanceId": "67bf0da2-29ed-4217-990c-d895d51812b9",
            "AcknowledgedAt": "2013-10-25T19:38:24+00:00",
            "LogUrl": "https://s3.amazonaws.com/prod_stage-log/logs/
b6c402df-5c23-45b2-a707-ad20b9c5ae40?AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
&Expires=1382731518&Signature=YkqS5IZN2P4wixjHwoC3aCMbn5s%3D&response-cache-
control=private&response-content-encoding=gzip&response-content-
type=text%2Fplain",
            "Type": "configure",
            "CommandId": "b6c402df-5c23-45b2-a707-ad20b9c5ae40",
            "CreatedAt": "2013-10-25T19:38:11+00:00",
            "ExitCode": 0
        },
        {
            "Status": "successful",
            "CompletedAt": "2013-10-25T19:31:08+00:00",
            "InstanceId": "67bf0da2-29ed-4217-990c-d895d51812b9",
            "AcknowledgedAt": "2013-10-25T19:29:01+00:00",
            "LogUrl": "https://s3.amazonaws.com/prod_stage-log/logs/2a90e862-
f974-42a6-9342-9a4f03468358?AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
&Expires=1382731518&Signature=cxKYHO8mCCd4MvOyFb6ywebeQtA%3D&response-cache-
control=private&response-content-encoding=gzip&response-content-
type=text%2Fplain",
            "Type": "setup",
            "CommandId": "2a90e862-f974-42a6-9342-9a4f03468358",
            "CreatedAt": "2013-10-25T19:26:01+00:00",
            "ExitCode": 0
        }
    ]
}
```


3. Copy the `LogUrl` value to your browser to view the log.

If the instance has more than a few commands, you can add parameters to `describe-commands` to filter which commands are included in the response object. For more information, see describe-commands.

Viewing a Chef Log on an Instance

> **Note**
>
> Topics in this section apply to Chef 12. For information about the location of Chef logs for Chef 11.10 and older releases, see Troubleshooting Chef 11.10 and Earlier Versions for Linux.

Linux instances

OpsWorks Stacks stores each instance's Chef logs in its `/var/chef/runs` directory. (For Linux instances, this directory also includes the associated data bags, stored as JSON-formatted files.) You need sudo privileges to access this directory. The log for each run is in a file named `chef.log` inside of the individual run's subdirectory.

OpsWorks Stacks stores its internal logs in the instance's `/var/log/aws/opsworks` folder. The information is usually not very helpful for troubleshooting purposes. However, these logs are useful to OpsWorks Stacks support, and you might be asked to provide them if you encounter an issue with the service. The Linux logs can also sometimes provide useful troubleshooting data.

Windows instances

Agent Logs

On Windows instances, OpsWorks logs are stored in a `ProgramData` path such as the following. The number includes a timestamp.

```
C:\ProgramData\OpsWorksAgent\var\logs\number
```

> **Note**
>
> By default, `ProgramData` is a hidden folder. To unhide it, navigate to **Folder Options**. Under **View**, choose the option to show hidden files.

The following example shows agent logs on a Windows instance.

```
Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---         5/24/2015  11:59 PM     127277 command.20150524.txt
-a---         5/25/2015  11:59 PM     546772 command.20150525.txt
-a---         5/26/2015  11:59 PM     551514 command.20150526.txt
-a---         5/27/2015   9:43 PM     495181 command.20150527.txt
-a---         5/24/2015  11:59 PM      24353 keepalive.20150524.txt
-a---         5/25/2015  11:59 PM     106232 keepalive.20150525.txt
-a---         5/26/2015  11:59 PM     106208 keepalive.20150526.txt
-a---         5/27/2015   8:54 PM      92593 keepalive.20150527.txt
-a---         5/24/2015   7:19 PM       3891 service.20150524.txt
-a---         5/27/2015   8:54 PM       1493 service.20150527.txt
-a---         5/24/2015  11:59 PM     112549 wire.20150524.txt
-a---         5/25/2015  11:59 PM     501501 wire.20150525.txt
-a---         5/26/2015  11:59 PM     499640 wire.20150526.txt
-a---         5/27/2015   8:54 PM     436870 wire.20150527.txt
```

Chef Logs

On Windows instances, Chef logs are stored in a `ProgramData` path such as the following. The number includes a timestamp.

```
C:\ProgramData\OpsWorksAgent\var\commands\number
```

> **Note**
>
> This directory contains only the output of the first (OpsWorks owned) Chef run.

The following example shows OpsWorks owned Chef logs on a Windows instance.

```
 Mode                LastWriteTime            Name
 ----                -------------            ----
 d----         5/24/2015   7:23 PM
 configure-7ecb5f47-7626-439b-877f-5e7cb40ab8be
 d----         5/26/2015   8:30 PM            configure-8e74223b-d15d-4372-aeea-
a87b428ffc2b
 d----         5/24/2015   6:34 PM            configure-
c3980a1c-3d08-46eb-9bae-63514cee194b
```

```
 d----         5/26/2015   8:32 PM            grant_remote_access-70dbf834-1bfa-4fce-
b195-e50e85402f4c
 d----         5/26/2015  10:30 PM            revoke_remote_access-1111fce9-843a-4b27-
b93f-ecc7c5e9e05b
 d----         5/24/2015   7:21 PM            setup-754ec063-8b60-4cd4-
b6d7-0e89d7b7aa78
 d----         5/26/2015   8:27 PM            setup-af5bed36-5afd-4115-
af35-5766f88bc039
 d----         5/24/2015   6:32 PM            setup-d8abeffa-24d4-414b-
bfb1-4ad07319f358
 d----         5/24/2015   7:13 PM            shutdown-c7130435-9b5c-4a95-
be17-6b988fc6cf9a
 d----         5/26/2015   8:25 PM
 sync_remote_users-64c79bdc-1f6f-4517-865b-23d2def4180c
 d----         5/26/2015   8:48 PM
 update_custom_cookbooks-2cc59a94-315b-414d-85eb-2bdea6d76c6a
```

User Chef Logs

The logs for your Chef runs can be found in files named `logfile.txt` in a folder that is named after the numbered Chef command, as in the following diagram.

C:/chef └── runs └── command-12345 ├── attribs.json ├── client.rb └── logfile.txt

Interpreting a Chef Log

The beginning of the log contains mostly internal Chef logging.

```
# Logfile created on Thu Oct 17 17:25:12 +0000 2013 by logger.rb/1.2.6
[2013-10-17T17:25:12+00:00] INFO: *** Chef 11.4.4 ***
[2013-10-17T17:25:13+00:00] DEBUG: Building node object for php-app1.localdomain
[2013-10-17T17:25:13+00:00] DEBUG: Extracting run list from JSON attributes provided on
 command line
[2013-10-17T17:25:13+00:00] INFO: Setting the run_list to
 ["opsworks_custom_cookbooks::load", "opsworks_custom_cookbooks::execute"] from JSON
[2013-10-17T17:25:13+00:00] DEBUG: Applying attributes from json file
[2013-10-17T17:25:13+00:00] DEBUG: Platform is amazon version 2013.03
[2013-10-17T17:25:13+00:00] INFO: Run List is [recipe[opsworks_custom_cookbooks::load],
 recipe[opsworks_custom_cookbooks::execute]]
[2013-10-17T17:25:13+00:00] INFO: Run List expands to [opsworks_custom_cookbooks::load,
 opsworks_custom_cookbooks::execute]
[2013-10-17T17:25:13+00:00] INFO: Starting Chef Run for php-app1.localdomain
[2013-10-17T17:25:13+00:00] INFO: Running start handlers
```

```
[2013-10-17T17:25:13+00:00] INFO: Start handlers complete.
[2013-10-17T17:25:13+00:00] DEBUG: No chefignore file found at /opt/aws/opsworks/
releases/20131015111601_209/cookbooks/chefignore no files will be ignored
[2013-10-17T17:25:13+00:00] DEBUG: Cookbooks to compile: ["gem_support", "packages",
 "opsworks_bundler", "opsworks_rubygems", "ruby", "ruby_enterprise", "dependencies",
 "opsworks_commons", "scm_helper", :opsworks_custom_cookbooks]
[2013-10-17T17:25:13+00:00] DEBUG: Loading cookbook gem_support's library file: /
opt/aws/opsworks/releases/20131015111601_209/cookbooks/gem_support/libraries/
current_gem_version.rb
[2013-10-17T17:25:13+00:00] DEBUG: Loading cookbook packages's library file: /opt/aws/
opsworks/releases/20131015111601_209/cookbooks/packages/libraries/packages.rb
[2013-10-17T17:25:13+00:00] DEBUG: Loading cookbook dependencies's library file: /
opt/aws/opsworks/releases/20131015111601_209/cookbooks/dependencies/libraries/
current_gem_version.rb
[2013-10-17T17:25:13+00:00] DEBUG: Loading cookbook opsworks_commons's library file: /
opt/aws/opsworks/releases/20131015111601_209/cookbooks/opsworks_commons/libraries/
activesupport_blank.rb
[2013-10-17T17:25:13+00:00] DEBUG: Loading cookbook opsworks_commons's library file: /
opt/aws/opsworks/releases/20131015111601_209/cookbooks/opsworks_commons/libraries/
monkey_patch_chefgem_resource.rb
...
```

This part of the file is useful largely to Chef experts. Note that the run list includes only two recipes, even though most commands involve many more. These two recipes handle the task of loading and executing all the other built-in and custom recipes.

The most interesting part of the file is typically at the end. If a run ends successfully, you should see something like the following:

```
...
[Tue, 11 Jun 2013 16:00:50 +0000] DEBUG: STDERR:
[Tue, 11 Jun 2013 16:00:50 +0000] DEBUG: ---- End output of /sbin/service mysqld
 restart ----
[Tue, 11 Jun 2013 16:00:50 +0000] DEBUG: Ran /sbin/service mysqld restart returned 0
[Tue, 11 Jun 2013 16:00:50 +0000] INFO: service[mysql]: restarted successfully
[Tue, 11 Jun 2013 16:00:50 +0000] INFO: Chef Run complete in 84.07096 seconds
[Tue, 11 Jun 2013 16:00:50 +0000] INFO: cleaning the checksum cache
[Tue, 11 Jun 2013 16:00:50 +0000] DEBUG: removing unused checksum cache file /var/chef/
cache/checksums/chef-file--tmp-chef-rendered-template20130611-4899-8wef7e-0
[Tue, 11 Jun 2013 16:00:50 +0000] DEBUG: removing unused checksum cache file /var/chef/
cache/checksums/chef-file--tmp-chef-rendered-template20130611-4899-1xpwyb6-0
```

```
[Tue, 11 Jun 2013 16:00:50 +0000] DEBUG: removing unused checksum cache file /var/chef/
cache/checksums/chef-file--etc-monit-conf
[Tue, 11 Jun 2013 16:00:50 +0000] INFO: Running report handlers
[Tue, 11 Jun 2013 16:00:50 +0000] INFO: Report handlers complete
[Tue, 11 Jun 2013 16:00:50 +0000] DEBUG: Exiting
```

> **Note**
>
> You can use the agent CLI to display the log's tail during or after the run. For more information, see Displaying Chef Logs.

If a recipe fails, you should look for an ERROR-level output, which will contain an exception followed by a Chef stack trace, such as the following:

```
...
Please report any problems with the /usr/scripts/mysqlbug script!

[  OK  ]
MySQL Daemon failed to start.
Starting mysqld:  [FAILED]STDERR: 130611 15:07:55 [Warning] The syntax '--log-slow-
queries' is deprecated and will be removed in a future release. Please use '--slow-
query-log'/'--slow-query-log-file' instead.
130611 15:07:56 [Warning] The syntax '--log-slow-queries' is deprecated and will be
 removed in a future release. Please use '--slow-query-log'/'--slow-query-log-file'
 instead.
---- End output of /sbin/service mysqld start ----

/opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/mixin/command.rb:184:in `handle_command_failures'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/mixin/command.rb:131:in `run_command'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/provider/service/init.rb:37:in `start_service'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/provider/service.rb:60:in `action_start'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/resource.rb:406:in `send'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/resource.rb:406:in `run_action'
```

```
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/runner.rb:53:in `run_action'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/runner.rb:89:in `converge'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/runner.rb:89:in `each'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/runner.rb:89:in `converge'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/resource_collection.rb:94:in `execute_each_resource'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/resource_collection/stepable_iterator.rb:116:in `call'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/resource_collection/stepable_iterator.rb:116:in
 `call_iterator_block'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/resource_collection/stepable_iterator.rb:85:in `step'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/resource_collection/stepable_iterator.rb:104:in `iterate'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/resource_collection/stepable_iterator.rb:55:in
 `each_with_index'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/resource_collection.rb:92:in `execute_each_resource'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/runner.rb:84:in `converge'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/client.rb:268:in `converge'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/client.rb:158:in `run'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/application/solo.rb:190:in `run_application'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/application/solo.rb:181:in `loop'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/application/solo.rb:181:in `run_application'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/../lib/chef/application.rb:62:in `run'
  /opt/aws/opsworks/releases/20130605160141_122/vendor/bundle/ruby/1.8/gems/
chef-0.9.15.5/bin/chef-solo:25
  /opt/aws/opsworks/current/bin/chef-solo:16:in `load'
  /opt/aws/opsworks/current/bin/chef-solo:16
```

The end of the file is the Chef stack trace. You should also examine the output just before the exception, which often contains a system error such as `package not available` that can also be useful in determining the failure cause. In this case, the MySQL daemon failed to start.

Common Chef Log Errors

The following are some common Chef log errors, and how to address them.

Log could not be found

At the start of a Chef run, instances receive a presigned Amazon S3 URL that lets you view the log on a webpage when the Chef run is finished. Because this URL expires after two hours, there is no log uploaded to the Amazon S3 site if a Chef run takes longer than two hours, even if no problems occurred during the Chef run. The command to create a log succeeds, but the log can be viewed only on the instance, not at the presigned URL.

Log ends abruptly

If a Chef log ends abruptly without either indicating success or displaying error information, you have probably encountered a low-memory state that prevented Chef from completing the log. Your best option is to try again with a larger instance.

Missing cookbook or recipe

If the Chef run encounters a cookbook or recipe that is not in the cookbook cache, you will see something like the following:

```
DEBUG: Loading Recipe mycookbook::myrecipe via include_recipe
ERROR: Caught exception during execution of custom recipe: mycookbook::myrecipe:
   Cannot find a cookbook named mycookbook; did you forget to add metadata to a
 cookbook?
```

This entry indicates that the `mycookbook` cookbook is not in the cookbook cache. With Chef 11.4, you can also encounter this error if you do not declare dependencies correctly in `metadata.rb`.

OpsWorks Stacks runs recipes from the instance's cookbook cache. It downloads cookbooks from your repository to this cache when the instance starts. However, OpsWorks Stacks does not automatically update the cache on an online instance if you subsequently modify the cookbooks in your repository. If you have modified your cookbooks or added new cookbooks since starting the instance, take the following steps:

1. Make sure that you have committed your changes to the repository.
2. Run the Update Cookbooks stack command to update the cookbook cache with the most recent version from the repository.

Local command failure

If a Chef `execute` resource fails to execute the specified command, you will see something like the following:

```
DEBUG: ---- End output of ./configure --with-config-file-path=/ returned 2
ERROR: execute[PHP: ./configure] (/root/opsworks-agent/site-cookbooks/php-fpm/
recipes/install.rb line 48) had an error:
   ./configure --with-config-file-path=/
```

Scroll up in the log and you should see the command's `stderr` and `stdout` output, which should help you determine why the command failed.

Package failure

If a package installation fails, you will see something like the following:

```
ERROR: package[zend-server-ce-php-5.3] (/root/opsworks-agent/site-cookbooks/
zend_server/recipes/install.rb line 20)
   had an error: apt-get -q -y --force-yes install zend-server-ce-php-5.3=5.0.4+b17
 returned 100, expected 0
```

Scroll up in the log and you should see the command's STDOUT and STDERROR output, which should help you determine why the package installation failed.

Using the OpsWorks Stacks Agent CLI

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

> **Note**
>
> The agent CLI is available only on Linux instances.

On every online instance, OpsWorks Stacks installs an agent, which communicates with the service. The OpsWorks Stacks service in turn sends commands to the agent to performs tasks such as initiating Chef runs on the instance when a lifecycle event occurs. On Linux instances, the agent exposes a command line interface (CLI) that is very useful for troubleshooting. To run agent CLI commands, use SSH to connect to an instance. You can then run agent CLI commands to perform a variety of tasks, including the following:

- Execute recipes.
- Display Chef logs.
- Display stack configuration and deployment JSON.

For more information on how to set up an SSH connection to an instance, see Logging In with SSH. You must also have SSH and sudo permissions for the stack.

This section describes how to use the agent CLI for troubleshooting. For more information and a complete command reference, see OpsWorks Stacks Agent CLI.

Topics

- Executing Recipes
- Displaying Chef Logs
- Displaying the Stack Configuration and Deployment JSON

Executing Recipes

The agent CLI `run_command` command directs the agent to rerun a command that it performed earlier. The most useful commands for troubleshooting—`setup`, `configure`, `deploy`, and `undeploy`—each correspond to a lifecycle event. They direct the agent to initiate a Chef run to execute the associated recipes.

> **Note**
>
> The `run_command` command is limited to executing the group of recipes that is associated with a specified command, typically the recipes that are associated with a lifecycle event. You cannot use it to execute a particular recipe. To execute one or more specified recipes, use the Execute Recipes stack command or the equivalent CLI or API actions (`create-deployment` and `CreateDeployment`).

The `run_command` command is quite useful for debugging custom recipes, especially recipes that are assigned to the Setup and Configure lifecycle events, which you can't trigger directly from the console. By using `run_command`, you can run a particular event's recipes as often as you need without having to start or stop instances.

> **Note**
>
> OpsWorks Stacks runs recipes from the instance's cookbook cache, not the cookbook repository. OpsWorks Stacks downloads cookbooks to this cache when the instance starts, but does not automatically update the cache on online instances if you subsequently modify your cookbooks. If you have modified your cookbooks since starting the instance, be sure to run the Update Cookbooks stack command stack command to update the cookbook cache with the most recent version from the repository.

The agent caches only the most recent commands. You can list them by running `list_commands`, which returns a list of cached commands and the time that they were performed.

```
sudo opsworks-agent-cli list_commands
2013-02-26T19:08:26        setup
2013-02-26T19:12:01        configure
2013-02-26T19:12:05        configure
2013-02-26T19:22:12        deploy
```

To rerun the most recent command, run this:

```
sudo opsworks-agent-cli run_command
```

To rerun the most recent instance of a specified command, run this:

```
sudo opsworks-agent-cli run_command command
```

For example, to rerun the Setup recipes, you can run the following command:

```
sudo opsworks-agent-cli run_command setup
```

Each command has an associated stack configuration and deployment JSON that represents stack and deployment state at the time the command was run. Because that data can change from one command to the next, an older instance of a command might use somewhat different data than the most recent one. To rerun a particular instance of a command, copy the time from the `list_commands` output and run the following:

```
sudo opsworks-agent-cli run_command time
```

The preceding examples all rerun the command using the default JSON, which is the JSON was installed for that command. You can rerun a command against an arbitrary JSON file as follows:

```
sudo opsworks-agent-cli run_command -f /path/to/valid/json.file
```

Displaying Chef Logs

The agent CLI `show_log` command displays a specified log. After the command is finished, you will be looking at the end of the file. The `show_log` command therefore provides a convenient way to tail the log, which is typically where you find error information. You can scroll up to see the earlier parts of the log.

To display the current command's log, run this:

```
sudo opsworks-agent-cli show_log
```

You can also display logs for a particular command, but be aware that the agent caches logs for only the last thirty commands. You can list an instance's commands by running `list_commands`, which returns a list of cached commands and the time that they were performed. For an example, see Executing Recipes.

To show the log for the most recent execution of a particular command, run the following:

```
sudo opsworks-agent-cli show_log command
```

The command parameter can be set to `setup`, `configure`, `deploy`, `undeploy`, `start`, `stop`, or `restart`. Most of these commands correspond to lifecycle events and direct the agent to run the associated recipes.

To display the log for a particular command execution, copy the date from the `list_commands` output and run:

```
sudo opsworks-agent-cli show_log date
```

If a command is still executing, `show_log` displays the log's current state.

> **Note**
>
> One way to use `show_log` to troubleshoot errors and out-of-memory issues is to tail a log during execution, as follows:
>
> 1. Use `run_command` to trigger the appropriate lifecycle event. For more information, see Executing Recipes.
> 2. Repeatedly run `show_log` to see the tail of the log as it is being written.
>
> If Chef runs out of memory or exits unexpectedly, the log will end abruptly. If a recipe fails, the log will end with an exception and a stack trace.

Displaying the Stack Configuration and Deployment JSON

Much of the data used by recipes comes from the stack configuration and deployment JSON, which defines a set of Chef attributes that provide a detailed description of the stack configuration, any deployments, and optional custom attributes that users can add. For each command, OpsWorks Stacks installs a JSON that represents the stack and deployment state at the time of the command . For more information, see Stack Configuration and Deployment Attributes.

If your custom recipes obtain data from the stack configuration and deployment JSON, you can verify the data by examining the JSON. The easiest way to display the stack configuration and deployment JSON is to run the agent CLI `get_json` command, which displays a formatted version of the JSON object. The following shows the first few lines of some typical output:

```
{
  "opsworks": {
    "layers": {
      "php-app": {
        "id": "4a2a56c8-f909-4b39-81f8-556536d20648",
        "instances": {
          "php-app2": {
            "elastic_ip": null,
            "region": "us-west-2",
            "booted_at": "2013-02-26T20:41:10+00:00",
            "ip": "10.112.235.192",
            "aws_instance_id": "i-34037f06",
            "availability_zone": "us-west-2a",
            "instance_type": "c1.medium",
            "private_dns_name": "ip-10-252-0-203.us-west-2.compute.internal",
            "private_ip": "10.252.0.203",
            "created_at": "2013-02-26T20:39:39+00:00",
            "status": "online",
            "backends": 8,
            "public_dns_name": "ec2-10-112-235-192.us-west-2.compute.amazonaws.com"
...
```

You can display the most recent stack configuration and deployment JSON as follows:

```
sudo opsworks-agent-cli get_json
```

You can display the most recent stack configuration and deployment JSON for a specified command by executing the following:

```
sudo opsworks-agent-cli get_json command
```

The command parameter can be set to `setup`, `configure`, `deploy`, `undeploy`, `start`, `stop`, or `restart`. Most of these commands correspond to lifecycle events and direct the agent to run the associated recipes.

You can display the stack configuration and deployment JSON for a particular command execution by specifying the command's date like this:

```
sudo opsworks-agent-cli get_json date
```

The simplest way to use this command is as follows:

1. Run `list_commands`, which returns a list of commands that have been run on the instance, and the date that each command was run.
2. Copy the date for the appropriate command and use it as the `get_json` *`date`* argument.

Common Debugging and Troubleshooting Issues

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section describes some commonly encountered debugging and troubleshooting issues and their solutions.

Topics

- Troubleshooting OpsWorks Stacks
- Troubleshooting Instance Registration

Troubleshooting OpsWorks Stacks

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

This section contains some commonly encountered OpsWorks Stacks issues and their solutions.

Topics

- Unable to Manage Instances
- After a Chef Run, Instances Won't Boot
- A Layer's Instances All Fail an Elastic Load Balancing Health Check
- Can't Communicate with an Elastic Load Balancing Load Balancer
- An Imported On-premises Instance Fails to Finish Volume Setup After a Restart
- An EBS Volume Does Not Reattach After a Reboot
- Can't Delete OpsWorks Stacks Security Groups
- Accidentally Deleted an OpsWorks Stacks Security Group
- Chef Log Terminates Abruptly
- Cookbook Does Not Get Updated
- Instances are Stuck at Booting Status
- Instances Unexpectedly Restart
- opsworks-agent Processes are Running on Instances
- Unexpected execute_recipes Commands

Unable to Manage Instances

Problem: You are no longer able to manage an instance that has been manageable in the past. In some cases, logs can show an error similar to the following.

```
Aws::CharlieInstanceService::Errors::UnrecognizedClientException - The security token
 included in the request is invalid.
```

Cause: This can occur if a resource outside OpsWorks on which the instance depends was edited or deleted. The following are examples of resource changes that can break communications with an instance.

- An IAM user or role associated with the instance has been deleted accidentally, outside of OpsWorks Stacks. This causes a communication failure between the OpsWorks agent that is installed on the instance, and the OpsWorks Stacks service. The user that is associated with an instance is required throughout the life of the instance.
- Editing volume or storage configurations while an instance is offline can make an instance unmanageable.

- Adding EC2 instances to an ELB manually. OpsWorks reconfigures an assigned Elastic Load Balancing load balancer each time an instance enters or leaves the online state. OpsWorks only considers instances it knows about to be valid members; instances that are added outside of OpsWorks, or by some other process, are removed. Every other instance is removed.

Solution: Do not delete IAM users or roles upon which your instances depend. If possible, edit volume or storage configurations only while dependent instances are running. Use OpsWorks to manage the load balancer or EIP memberships of OpsWorks instances. When you are registering an instance, to help prevent problems managing registered instances in the event that the user is accidentally deleted, add the `--use-instance-profile` parameter to your `register` command to use the instance's built-in instance profile instead.

After a Chef Run, Instances Won't Boot

Problem: On Chef 11.10 or older stacks that are configured to use custom cookbooks, after a Chef run that used community cookbooks, instances won't boot. Log messages can state that recipes failed to compile ("Recipe Compile Error"), or can't be loaded because they are unable to find a dependency.

Cause: The most likely cause is that the custom or community cookbook does not support the Chef version that your stack uses. Some popular community cookbooks, such as `apt` and `build-essential`, have known compatibility issues with Chef 11.10.

Solution: On OpsWorks Stacks stacks that have the **Use custom Chef cookbooks** setting turned on, custom or community cookbooks must always support the version of Chef that your stack uses. Pin community cookbooks to a version (that is, set the cookbook version number to a specific version) that is compatible with the version of Chef that is configured in your stack settings. To find a supported community cookbook version, view the changelog for a cookbook that fails to compile, and use only the most current version of the cookbook that your stack will support. To pin a cookbook version, specify an exact version number in your custom cookbook repository's Berksfile. For example, `cookbook 'build-essential', '= 3.2.0'`.

A Layer's Instances All Fail an Elastic Load Balancing Health Check

Problem: You attach an Elastic Load Balancing load balancer to an app server layer, but all the instances fail the health check.

Cause: When you create an Elastic Load Balancing load balancer, you must specify the ping path that the load balancer calls to determine whether the instance is healthy. Be sure to specify a ping

path that is appropriate for your application; the default value is /index.html. If your application does not include an `index.html`, you must specify an appropriate path or the health check will fail. For example, the SimplePHPApp application used in Getting Started with Chef 11 Linux Stacks does not use index.html; the appropriate ping path for those servers is /.

Solution: Edit the load balancer's ping path. For more information, see Elastic Load Balancing

Can't Communicate with an Elastic Load Balancing Load Balancer

Problem: You create an Elastic Load Balancing load balancer and attach it to an app server layer, but when you click the load balancer's DNS name or IP address to run the application, you get the following error: "The remote server is not responding".

Cause: If your stack is running in a default VPC, when you create an Elastic Load Balancing load balancer in the region, you must specify a security group. The security group must have ingress rules that allow inbound traffic from your IP address. If you specify **default VPC security group**, the default ingress rule does not accept any inbound traffic.

Solution: Edit the security group ingress rules to accept inbound traffic from appropriate IP addresses.

1. Click **Security Groups** in the Amazon EC2 console's navigation pane.
2. Select the load balancer's security group.
3. Click **Edit** on the **Inbound** tab.
4. Add an ingress rule with **Source** set to an appropriate CIDR.

   For example, specifying **Anywhere** sets the CIDR to 0.0.0.0/0, which directs the load balancer to accept incoming traffic from any IP address.

An Imported On-premises Instance Fails to Finish Volume Setup After a Restart

Problem: You restart an EC2 instance that you have imported into OpsWorks Stacks, and the OpsWorks Stacks console displays **failed** as the instance status. This can occur on either Chef 11 or Chef 12 instances.

Cause: OpsWorks Stacks might be unable to attach a volume to your instance during the setup process. One possible cause is that OpsWorks Stacks overwrites your volume configuration on your instance when you run the `setup` command.

Solution: Open the **Details** page for the instance, and check your volume configuration in the **Volumes** area. Note that you can change your volume configuration only when your instance is in the **stopped** state. Be sure that every volume has a specified mount point and a name. Confirm that you provided the correct mount point in your configuration in OpsWorks Stacks before you restart the instance.

An EBS Volume Does Not Reattach After a Reboot

Problem: You use the Amazon EC2 console to attach an Amazon EBS volume to an instance but when you reboot the instance, the volume is no longer attached.

Cause: OpsWorks Stacks can reattach only those Amazon EBS volumes that it is aware of, which are limited to the following:

- Volumes that were created by OpsWorks Stacks.
- Volumes from your account that you have explicitly registered with a stack by using the **Resources** page.

Solution: Manage your Amazon EBS volumes only by using the OpsWorks Stacks console, API, or CLI. If you want to use one of your account's Amazon EBS volumes with a stack, use the stack's **Resources** page to register the volume and attach it to an instance. For more information, see Resource Management.

Can't Delete OpsWorks Stacks Security Groups

Problem: After you delete a stack, there are a number of OpsWorks Stacks security groups left behind that can't be deleted.

Cause: The security groups must be deleted in a particular order.

Solution: First, make sure that no instances are using the security groups. Then, delete any of the following security groups, if they exist, in the following order:

1. AWS-OpsWorks-Blank-Server
2. AWS-OpsWorks-Monitoring-Master-Server
3. AWS-OpsWorks-DB-Master-Server
4. AWS-OpsWorks-Memcached-Server
5. AWS-OpsWorks-Custom-Server

6. AWS-OpsWorks-nodejs-App-Server
7. AWS-OpsWorks-PHP-App-Server
8. AWS-OpsWorks-Rails-App-Server
9. AWS-OpsWorks-Web-Server
10. AWS-OpsWorks-Default-Server
11. AWS-OpsWorks-LB-Server

Accidentally Deleted an OpsWorks Stacks Security Group

Problem: You deleted one of the OpsWorks Stacks security groups and need to recreate it.

Cause: These security groups are usually deleted by accident.

Solution: The recreated group must an exact duplicate of the original, including the same capitalization for the group name. Instead of recreating the group manually, the preferred approach is to have OpsWorks Stacks perform the task for you. Just create a new stack in the same AWS region—and VPC, if present—and OpsWorks Stacks will automatically recreate all built-in security groups, including the one that you deleted. You can then delete the stack if you don't have any further use for it; the security groups will remain.

Chef Log Terminates Abruptly

Problem: A Chef log terminates abruptly; the end of the log does not indicate a successful run or display an exception and stack trace.

Cause: This behavior is typically caused by inadequate memory.

Solution: Create a larger instance and use the agent CLI `run_command` command to run the recipes again. For more information, see Executing Recipes.

Cookbook Does Not Get Updated

Problem: You updated your cookbooks but the stack's instances are still running the old recipes.

Cause: OpsWorks Stacks caches cookbooks on each instance, and runs recipes from the cache, not the repository. When you start a new instance, OpsWorks Stacks downloads your cookbooks from the repository to the instance's cache. However, if you subsequently modify your custom cookbooks, OpsWorks Stacks does not automatically update the online instances' caches.

Solution: Run theUpdate Cookbooks stack command to explicitly direct OpsWorks Stacks to update your online instances' cookbook caches.

Instances are Stuck at Booting Status

Problem: When you restart an instance, or auto healing restarts it automatically, the startup operation stalls at the `booting` status.

Cause: One possible cause of this issue is the VPC configuration, including a default VPC. Instances must always be able to communicate with the OpsWorks Stacks service, Amazon S3, and the package, cookbook, and app repositories. If, for example, you remove a default gateway from a default VPC, the instances lose their connection to the OpsWorks Stacks service. Because OpsWorks Stacks can no longer communicate with the instance agent, it treats the instance as failed and auto heals it. However, without a connection, OpsWorks Stacks cannot install an instance agent on the healed instance. Without an agent, OpsWorks Stacks cannot run the Setup recipes on the instance, so the startup operation cannot progress beyond the "booting" status.

Solution: Modify your VPC configuration so that instances have the required connectivity.

Instances Unexpectedly Restart

Problem: A stopped instance unexpectedly restarts.

Cause 1: If you have enabled auto healing for your instances, OpsWorks Stacks periodically performs a health check on the associated Amazon EC2 instances, and restarts any that are unhealthy. If you stop or terminate an OpsWorks Stacks-managed instance by using the Amazon EC2 console, API, or CLI, OpsWorks Stacks is not notified. Instead, it will perceive the stopped instance as unhealthy and automatically restart it.

Solution: Manage your instances only by using the OpsWorks Stacks console, API, or CLI. If you use OpsWorks Stacks to stop or delete an instance, it will not be restarted. For more information, see Manually Starting, Stopping, and Rebooting 24/7 Instances and Deleting OpsWorks Stacks Instances.

Cause 2: Instances can fail for a variety of reasons. If you have auto healing enabled, OpsWorks Stacks automatically restarts failed instances.

Solution: This is normal operation; there is no need to do anything unless you do not want OpsWorks Stacks to restart failed instances. In that case, you should disable auto healing.

`opsworks-agent` Processes are Running on Instances

Problem: Several `opsworks-agent` processes are running on your instances. For example:

```
aws 24543 0.0 1.3 172360 53332 ? S Feb24 0:29 opsworks-agent: master 24543
aws 24545 0.1 2.0 208932 79224 ? S Feb24 22:02 opsworks-agent: keep_alive of master
 24543
aws 24557 0.0 2.0 209012 79412 ? S Feb24 8:04 opsworks-agent: statistics of master
 24543
aws 24559 0.0 2.2 216604 86992 ? S Feb24 4:14 opsworks-agent: process_command of master
 24
```

Cause: These are legitimate processes that are required for the agent's normal operation. They perform tasks such as handling deployments and sending keep-alive messages back to the service.

Solution: This is normal behavior. Do not stop these processes; doing so will compromise the agent's operation.

Unexpected execute_recipes Commands

Problem: The **Logs** section on an instance's details page includes unexpected `execute_recipes` commands. Unexpected `execute_recipes` commands can also appear on the **Stack** and **Deployments** pages.

Cause: This issue is often caused by permission changes. When you change a user or group's SSH or sudo permissions, OpsWorks Stacks runs `execute_recipes` to update the stack's instances and also triggers a Configure event. Another source of `execute_recipes` commands is OpsWorks Stacks updating the instance agent.

Solution: This is normal operation; there is no need to do anything.

To see what actions an `execute_recipes` command performed, go to the **Deployments** page and click the command's time stamp. This opens the command's details page, which lists the key recipes that were run. For example, the following details page is for an `execute_recipes` command that ran `ssh_users` to update SSH permissions.



To see all of the details, click **show** in the command's **Log** column to display the associated Chef log. Search the log for **`Run List`**. OpsWorks Stacks maintenance recipes will be under `OpsWorks Custom Run List`. For example, the following is the run list for the `execute_recipes` command shown in the preceding screenshot, and shows every recipe that is associated with the command.

```
[2014-02-21T17:16:30+00:00] INFO: OpsWorks Custom Run List:
 ["opsworks_stack_state_sync",
  "ssh_users", "test_suite", "opsworks_cleanup"]
```

Troubleshooting Instance Registration

This section contains some commonly encountered instance registration issues and their solutions.

> **Note**
>
> If you are having registration problems, run `register` with the `--debug` argument, which provides additional debugging information.

Topics

- EC2User Is Not Authorized to Perform: ...
- Credential Should Be Scoped to a Valid Region

EC2User Is Not Authorized to Perform: ...

Problem: A `register` command returns something like the following:

```
A client error (AccessDenied) occurred when calling the CreateGroup operation:
User: arn:aws:iam::123456789012:user/ImportEC2User is not authorized to
perform: iam:CreateGroup on resource:
arn:aws:iam::123456789012:group/AWS/OpsWorks/OpsWorks-b583ce55-1d01-4695-b3e5-
ee19257d1911
```

Cause: The `register` command is running with credentials that do not grant the required permissions. The user's policy must allow the `iam:CreateGroup` action, among others.

Solution Provide `register` with IAM user credentials that have the required permissions. For more information, see Installing and Configuring the AWS CLI.

Credential Should Be Scoped to a Valid Region

Problem: A `register` command returns the following:

```
A client error (InvalidSignatureException) occurred when calling the
DescribeStacks operation: Credential should be scoped to a valid region, not 'cn-
north-1'.
```

Cause: The command's region must be a valid OpsWorks Stacks region. For a list of supported regions, see Region Support. This error typically occurs for one of the following reasons:

- The stack is in a different region, and you assigned a the stack's region to the command's `--region` argument.

  You don't need to specify a stack region; OpsWorks Stacks automatically determines it from the stack ID.
- You omitted `--region` argument, which implicitly specifies the default region, but your default region is not supported by OpsWorks Stacks.

Solution: Explicitly set `--region` to a supported OpsWorks Stacks region, or edit your AWS CLI `config` file to change the default region to a supported OpsWorks Stacks region. For more information, see Configuring the AWS Command Line Interface.

OpsWorks Stacks Agent CLI

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Note

This feature is available only on Linux instances.

The agent that OpsWorks Stacks installs on every instance exposes a command line interface (CLI). If you use SSH to log in to the instance, you can use the CLI to the following:

- Access log files for Chef runs.
- Access OpsWorks Stacks commands.
- Manually run Chef recipes.
- View instance reports.
- View agent reports.
- View a limited set of stack configuration and deployment attributes.

Important

You can run agent CLI commands only as root or by using sudo.

The basic command syntax is:

```
sudo opsworks-agent-cli [--help] [command [activity] [date]]
```

The four arguments are as follows:

help

(Optional) Displays a brief synopsis of the available commands when used by itself. When used with a command, `help` displays a description of the command.

command

(Optional) The agent CLI command, which must be set to one of the following:

- agent_report
- get_json
- instance_report
- list_commands
- run_command
- show_log
- stack_state

activity

(Optional) Used as an argument with some commands to specify a particular OpsWorks Stacks activity: `setup`, `configure`, `deploy`, `undeploy`, `start`, `stop`, or `restart`.

date

(Optional) Used as an argument with some commands to specify a particular OpsWorks Stacks command execution. Specify the command execution by setting **date** to the timestamp that the command was executed in the *`yyyy-mm-ddThh:mm:ss`* format, including the single quotes. For example, for 10:31:55 on Tuesday Feb 5, 2013, use: `'2013-02-05T10:31:55'`. To determine when a particular OpsWorks Stacks command was executed, run list_commands.

> **Note**
>
> If the agent has executed the same OpsWorks Stacks activity multiple times, you can pick a particular execution by specifying both the activity and the time it was executed. If you specify an activity and omit the time, the agent CLI command acts on the activity's most recent execution. If you omit both arguments, the agent CLI command acts on the most recent activity.

The following sections describe the commands and their associated arguments. For brevity, the syntax sections omit the optional `--help` option, which can be used with any command.

Topics

- agent_report
- get_json
- instance_report
- list_commands
- run_command
- show_log
- stack_state

agent_report

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Returns an agent report.

```
sudo opsworks-agent-cli agent_report
```

The following output example is from an instance that most recently ran a configure activity.

```
$ sudo opsworks-agent-cli agent_report

AWS OpsWorks Instance Agent State Report:

  Last activity was a "configure" on 2015-12-01 18:19:23 UTC
  Agent Status: The AWS OpsWorks agent is running as PID 30998
  Agent Version: 4004-20151201152533, up to date
```

get_json

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Returns information about a Chef run as a JSON object.

```
sudo opsworks-agent-cli get_json [activity] [date] [-i | --internal | --no-i | --no-internal]
```

By default, `get_json` displays customer-supplied information for the most recent Chef run. Use the following options to specify a particular set of information.

activity

Displays information for the Chef run associated with the most recent specified activity. To get a list of valid activities, run `list_commands`.

date

Displays information for the Chef run associated with the activity that executed for the specified timestamp. To get a list of valid timestamps, run `list_commands`.

-i, --internal

Displays information that OpsWorks Stacks uses internally for the Chef run.

--no-i, --no-internal

Explicitly displays customer-supplied information for the Chef run. This is the default if not otherwise specified.

> **Note**
>
> For Chef 12 Linux instances, running this command will return valid information such as the instance's stack configuration and deployment attributes. However, to get more complete information, reference the Chef data bags that OpsWorks Stacks creates on the instance. For more information, see the OpsWorks Stacks Data Bag Reference.

The following output example shows the customer-supplied information for the most recent Chef run for the most recent configure activity.

```
$ sudo opsworks-agent-cli get_json configure

{
  "run_list": [
    "recipe[opsworks_cookbook_demo::configure]"
  ]
}
```

The following output example shows information that OpsWorks Stacks uses internally for the Chef run executed for the specified timestamp.

```
$ sudo opsworks-agent-cli get_json 2015-12-01T18:20:24 -i

{
  "aws_opsworks_agent": {
    "version": "4004-20151201152533",
    "valid_client_activities": [
      "reboot",
      "stop",
      "deploy",
      "grant_remote_access",
      "revoke_remote_access",
      "update_agent",
      "setup",
      "configure",
      "update_dependencies",
      "install_dependencies",
      "update_custom_cookbooks",
      "execute_recipes",
      "sync_remote_users"
```

```
    ],
    "command": {
      "type": "configure",
      "args": {
        "app_ids": [

        ]
      },
      "sent_at": "2015-12-01T18:19:23+00:00",
      "command_id": "5c2113f3-c6d5-40eb-bcfa-77da2885eeEX",
      "iam_user_arn": null,
      "instance_id": "cfdaa716-42fe-4e3b-9762-fef184ddd8EX"
    },
    "resources": {
      "apps": [

      ],
      "layers": [
        {
          "layer_id": "93f50d83-1e73-45c4-840a-0d4f07cda1EX",
          "name": "MyCookbooksDemoLayer",
          "packages": [

          ],
          "shortname": "cookbooks-demo",
          "type": "custom",
          "volume_configurations": [

          ]
        }
      ],
      "instances": [
        {
          "ami_id": "ami-d93622EX",
          "architecture": "x86_64",
          "auto_scaling_type": null,
          "availability_zone": "us-west-2a",
          "created_at": "2015-11-18T00:21:05+00:00",
          "ebs_optimized": false,
          "ec2_instance_id": "i-a480e960",
          "elastic_ip": null,
          "hostname": "cookbooks-demo1",
          "instance_id": "cfdaa716-42fe-4e3b-9762-fef184ddd8EX",
          "instance_type": "c3.large",
```

```
          "layer_ids": [
            "93f50d83-1e73-45c4-840a-0d4f07cda1EX"
          ],
          "os": "Amazon Linux 2015.09",
          "private_dns": "ip-192-0-2-0.us-west-2.compute.internal",
          "private_ip": "10.122.69.33",
          "public_dns": "ec2-203-0-113-0.us-west-2.compute.amazonaws.com",
          "public_ip": "192.0.2.0",
          "root_device_type": "ebs",
          "root_device_volume_id": "vol-f6f7e8EX",
          "ssh_host_dsa_key_fingerprint": "f2:...:15",
          "ssh_host_dsa_key_public": "ssh-dss AAAAB3Nz...a8vMbqA=",
          "ssh_host_rsa_key_fingerprint": "0a:...:96",
          "ssh_host_rsa_key_public": "ssh-rsa AAAAB3Nz...yhPanvo7",
          "status": "online",
          "subnet_id": null,
          "virtualization_type": "paravirtual",
          "infrastructure_class": "ec2",
          "ssh_host_dsa_key_private": "-----BEGIN DSA PRIVATE KEY-----
\nMIIDVwIB...g5OtgQ==\n-----END DSA PRIVATE KEY-----\n",
          "ssh_host_rsa_key_private": "-----BEGIN RSA PRIVATE KEY-----
\nMIIEowIB...78kprtIw\n-----END RSA PRIVATE KEY-----\n"
        }
      ],
      "users": [

      ],
      "elastic_load_balancers": [

      ],
      "rds_db_instances": [

      ],
      "stack": {
        "arn": "arn:aws:opsworks:us-west-2:80398EXAMPLE:stack/040c3def-b2b4-4489-bb1b-
e08425886fEX/",
        "custom_cookbooks_source": {
          "type": "s3",
          "url": "https://s3.amazonaws.com/amzn-s3-demo-bucket/opsworks-cookbook-
demo.tar.gz",
          "username": "AKIAJUQN...WG644EXA",
          "password": "O5v+4Zz+...rcKbFTJu",
          "ssh_key": null,
          "revision": null
```

```
        },
        "name": "MyCookbooksDemoStack",
        "region": "us-west-2",
        "stack_id": "040c3def-b2b4-4489-bb1b-e08425886fEX",
        "use_custom_cookbooks": true,
        "vpc_id": null
      },
      "ecs_clusters": [

      ],
      "volumes": [

      ]
    },
    "chef": {
      "customer_recipes": [
        "opsworks_cookbook_demo::configure"
      ],
      "customer_json": "e30=\n",
      "customer_data_bags": "e30=\n"
    }
  }
}
```

instance_report

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Returns an extended instance report.

```
sudo opsworks-agent-cli instance_report
```

The following output example is from an instance.

```
$ sudo opsworks-agent-cli instance_report

AWS OpsWorks Instance Agent State Report:

  Last activity was a "configure" on 2015-12-01 18:19:23 UTC
  Agent Status: The AWS OpsWorks agent is running as PID 30998
  Agent Version: 4004-20151201152533, up to date
  OpsWorks Stack: MyCookbooksDemoStack
  OpsWorks Layers: MyCookbooksDemoLayer
  OpsWorks Instance: cookbooks-demo1
  EC2 Instance ID: i-a480e9EX
  EC2 Instance Type: c3.large
  Architecture: x86_64
  Total Memory: 3.84 Gb
  CPU: 2x  Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz

Location:

  EC2 Region: us-west-2
  EC2 Availability Zone: us-west-2a

Networking:

  Public IP: 192.0.2.0
  Private IP: 198.51.100.0
```

list_commands

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Lists the times for each activity that has been executed on this instance. You can use these times for other agent-CLI commands to specify a particular execution.

```
sudo opsworks-agent-cli list_commands  [activity] [date]
```

The following output example is from an instance that has run configure, setup, and update custom cookbooks activities.

```
$ sudo opsworks-agent-cli list_commands

2015-11-24T21:00:28        update_custom_cookbooks
2015-12-01T18:19:09        setup
2015-12-01T18:20:24        configure
```

run_command

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Runs an OpsWorks Stacks command, which is a JSON file containing a Chef run-list that contains the information necessary to execute an OpsWorks Stacks activity (setup, configure, deploy, and so on). The `run_command` command generates a log entry that you can view by running `show_log`. This option is intended only for development purposes, so OpsWorks Stacks does not track changes.

```
sudo opsworks-agent-cli run_command [activity] [date] [/path/to/valid/json.file]
```

By default, `run_command` runs the most recent OpsWorks Stacks command. Use the following options to specify a particular command.

activity

Run a specified OpsWorks Stacks command: `setup`, `configure`, `deploy`, `undeploy`, `start`, `stop`, or `restart`.

date

Run the AWS OpsWorks command that executed at the specified timestamp. To get a list of valid timestamps, run list_commands.

file

Run the specified command JSON file. To get a command's file path, run get_json.

The following output example is from an instance and runs the configure command.

```
$ sudo opsworks-agent-cli run_command configure

[2015-12-02 16:52:53]  INFO [opsworks-agent(21970)]: About to re-run 'configure' from
 2015-12-01T18:20:24
...
[2015-12-02 16:53:02]  INFO [opsworks-agent(21970)]: Finished Chef run with exitcode 0
```

show_log

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Return's a command's log file.

```
sudo opsworks-agent-cli show_log [activity] [date]
```

By default, `show_log` tails the most recent log file. Use the following options to specify a particular command.

activity

Display the specified activity's log file.

date

Display the log file for the activity that executed at the specified timestamp. To get a list of valid timestamps, run list_commands.

The following output example shows the most recent log.

```
$ sudo opsworks-agent-cli show_log

[2015-12-02T16:52:59+00:00] INFO: Storing updated cookbooks/opsworks_cookbook_demo/
opsworks-cookbook-demo.tar.gz in the cache.
...
[2015-12-02T16:52:59+00:00] INFO: Report handlers complete
```

stack_state

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Displays information that OpsWorks Stacks uses internally for the most recent Chef run.

```
opsworks-agent-cli stack_state
```

Note

For Chef 12 Linux instances, running this command will return valid information such as the instance's stack configuration and deployment attributes. However, to get more complete information, reference the Chef data bags that OpsWorks Stacks creates on the instance. For more information, see the OpsWorks Stacks Data Bag Reference.

The following output example is from an instance.

```
$ sudo opsworks-agent-cli stack_state

{
  "last_command": {
    "sent_at": "2015-12-01T18:19:23+00:00",
    "activity": "configure"
  },
  "instance": {
    "ami_id": "ami-d93622EX",
    "architecture": "x86_64",
    "auto_scaling_type": null,
    "availability_zone": "us-west-2a",
    "created_at": "2015-11-18T00:21:05+00:00",
    "ebs_optimized": false,
    "ec2_instance_id": "i-a480e9EX",
    "elastic_ip": null,
    "hostname": "cookbooks-demo1",
    "instance_id": "cfdaa716-42fe-4e3b-9762-fef184ddd8EX",
    "instance_type": "c3.large",
    "layer_ids": [
      "93f50d83-1e73-45c4-840a-0d4f07cda1EX"
    ],
    "os": "Amazon Linux 2015.09",
    "private_dns": "ip-192-0-2-0.us-west-2.compute.internal",
    "private_ip": "10.122.69.33",
    "public_dns": "ec2-203-0-113-0.us-west-2.compute.amazonaws.com",
    "public_ip": "192.0.2.0",
    "root_device_type": "ebs",
    "root_device_volume_id": "vol-f6f7e8EX",
    "ssh_host_dsa_key_fingerprint": "f2:...:15",
    "ssh_host_dsa_key_public": "ssh-dss AAAAB3Nz...a8vMbqA=",
    "ssh_host_rsa_key_fingerprint": "0a:...:96",
    "ssh_host_rsa_key_public": "ssh-rsa AAAAB3Nz...yhPanvo7",
    "status": "online",
    "subnet_id": null,
    "virtualization_type": "paravirtual",
    "infrastructure_class": "ec2",
    "ssh_host_dsa_key_private": "-----BEGIN DSA PRIVATE KEY-----\nMIIDVwIB...g5OtgQ==
\n-----END DSA PRIVATE KEY-----\n",
    "ssh_host_rsa_key_private": "-----BEGIN RSA PRIVATE KEY-----\nMIIEowIB...78kprtIw
\n-----END RSA PRIVATE KEY-----\n"
  },
  "layers": [
```

```
    {
      "layer_id": "93f50d83-1e73-45c4-840a-0d4f07cda1EX",
      "name": "MyCookbooksDemoLayer",
      "packages": [

      ],
      "shortname": "cookbooks-demo",
      "type": "custom",
      "volume_configurations": [

      ]
    }
  ],
  "applications": null,
  "stack": {
    "arn": "arn:aws:opsworks:us-west-2:80398EXAMPLE:stack/040c3def-b2b4-4489-bb1b-
e08425886fEX/",
    "custom_cookbooks_source": {
      "type": "s3",
      "url": "https://s3.amazonaws.com/amzn-s3-demo-bucket/opsworks-cookbook-
demo.tar.gz",
      "username": "AKIAJUQN...WG644EXA",
      "password": "O5v+4Zz+...rcKbFTJu",
      "ssh_key": null,
      "revision": null
    },
    "name": "MyCookbooksDemoStack",
    "region": "us-west-2",
    "stack_id": "040c3def-b2b4-4489-bb1b-e08425886fEX",
    "use_custom_cookbooks": true,
    "vpc_id": null
  },
  "agent": {
    "valid_activities": [
      "reboot",
      "stop",
      "deploy",
      "grant_remote_access",
      "revoke_remote_access",
      "update_agent",
      "setup",
      "configure",
      "update_dependencies",
      "install_dependencies",
```

```
      "update_custom_cookbooks",
      "execute_recipes",
      "sync_remote_users"
    ]
  }
}
```

OpsWorks Stacks Data Bag Reference

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

OpsWorks Stacks exposes a wide variety of settings to recipes as Chef data bag content. This reference lists this data bag content.

A *data bag* is a Chef concept. A data bag is a global variable that is stored as JSON data on an instance; the JSON data is accessible from Chef. For example, a data bag can store global variables such as an app's source URL, the instance's hostname, and the associated stack's VPC identifier. OpsWorks Stacks stores its data bags on each stack's instances. On Linux instances, OpsWorks Stacks stores data bags in the `/var/chef/runs/`*`run-ID`*`/data_bags` directory. On Windows instances, it stores data bags in the *`drive`*`:\chef\runs\`*`run-id`*`\data_bags` directory. In both cases, *`run-ID`* is a unique ID that OpsWorks Stacks assigns to each Chef run on an instance. These directories include a set of data bags (subdirectories). Each data bag contains zero or more data bag items, which are JSON-formatted files that contain sets of data bag content.

Note

OpsWorks Stacks does not support encrypted data bags. To store sensitive data in encrypted form, such as passwords or certificates, we recommend storing it in a private S3 bucket. You can then create a custom recipe that uses the Amazon SDK for Ruby to retrieve the data. For an example, see Using the SDK for Ruby.

Data bag content can include any of the following:

- **String** content that follows standard Ruby syntax and can use single or double quotes, although strings containing certain special characters must have double quotes. For more information, go to the Ruby documentation site.
- **Boolean** content, which is either `true` or `false` (no quotes).
- **Number** content, which is either integer or decimal numbers, such as `4` or `2.5` (no quotes).
- **List** content, which takes the form of comma-separated values enclosed in square brackets (no quotes), such as `[ '80', '443' ]`
- **JSON objects**, which contain additional data bag content, such as `"my-app": {"elastic_ip": null,...}`.

Chef recipes can access data bags, data bag items, and data bag content through Chef search or directly. The following describes how to use both access approaches (although Chef search is preferred).

To access a data bag through Chef search, use the search method, specifying the desired search index. OpsWorks Stacks provides the following search indexes:

- aws_opsworks_app, which represents a set of deployed apps for a stack.
- aws_opsworks_command, which represents a set of commands that were run on a stack.
- aws_opsworks_ecs_cluster, which represents a set of Amazon Elastic Container Service (Amazon ECS) cluster instances for a stack.
- aws_opsworks_elastic_load_balancer, which represents a set of Elastic Load Balancing load balancers for a stack.
- aws_opsworks_instance, which represents a set of instances for a stack.
- aws_opsworks_layer, which represents a set of layers for a stack.
- aws_opsworks_rds_db_instance, which represents a set of Amazon Relational Database Service (Amazon RDS) instances for a stack.
- aws_opsworks_stack, which represents a stack.
- aws_opsworks_user, which represents a set of users for a stack.

Once you know the search index name, you can access the content of the data bag for that search index. For example, the following recipe code uses the `aws_opsworks_app` search index to get

the content of the first data bag item (the first JSON file) in the`aws_opsworks_app` data bag (the `aws_opsworks_app` directory). The code then writes two messages to the Chef log, one with the app's shortname data bag content (a string in the JSON file), and another with the app's source URL data bag content (another string in the JSON file):

```
app = search("aws_opsworks_app").first
Chef::Log.info("********** The app's short name is '#{app['shortname']}' **********")
Chef::Log.info("********** The app's URL is '#{app['app_source']['url']}' **********")
```

Where `['shortname']` and `['app_source']['url']` specify the following data bag content in the corresponding JSON file:

```
{
  ...
  "shortname": "mylinuxdemoapp",
  ...
  "app_source": {
    ...
    "url": "https://s3.amazonaws.com/opsworks-demo-assets/opsworks-linux-demo-
nodejs.tar.gz",
  },
  ...
}
```

For a list of the data bag content that you can search for, see the reference topics in this section.

You can also iterate through a set of data bag items in a data bag. For example, the following recipe code is similar to the previous example; it iterates through each of the data bag items in the data bag when there is more than one data bag item:

```
search("aws_opsworks_app").each do |app|
  Chef::Log.info("********** The app's short name is '#{app['shortname']}' **********")
  Chef::Log.info("********** The app's URL is '#{app['app_source']['url']}'
 **********")
end
```

If you know that specific data bag content exists, you can find the corresponding data bag item with the following syntax:

```
search("search_index", "key:value").first
```

For example, the following recipe code uses the `aws_opsworks_app` search index to find the data bag item that contains the app short name of `mylinuxdemoapp`. It then uses the data bag item's contents to write a message to the Chef log with the corresponding app's short name and source URL:

```
app = search("aws_opsworks_app", "shortname:mylinuxdemoapp").first
Chef::Log.info("********** For the app with the short name '#{app['shortname']}', the
 app's URL is '#{app['app_source']['url']}' **********")
```

For the `aws_opsworks_instance` search index only, you can specify `self:true` to represent the instance that the recipe is being executed on. The following recipe code uses the corresponding data bag item's contents to write a message to the Chef log with the corresponding instance's OpsWorks Stacks-generated ID and operating system:

```
instance = search("aws_opsworks_instance", "self:true").first
Chef::Log.info("********** For instance '#{instance['instance_id']}', the instance's
 operating system is '#{instance['os']}' **********")
```

Instead of using Chef search to access data bags, data bag items, and data bag content, you can access them directly. To do this, use the data_bag and data_bag_item methods to access data bags and data bag items, respectively. For example, the following recipe code does the same things as the previous examples, except that it directly accesses a single data bag item and then multiple data bag items when there are more than one:

```
# Syntax: data_bag_item("the data bag name", "the file name in the data bag without the
 file extension")
app = data_bag_item("aws_opsworks_app", "mylinuxdemoapp")
Chef::Log.info("********** The app's short name is '#{app['shortname']}' **********")
Chef::Log.info("********** The app's URL is '#{app['app_source']['url']}' **********")

data_bag("aws_opsworks_app").each do |data_bag_item|
  app = data_bag_item("aws_opsworks_app", data_bag_item)
  Chef::Log.info("********** The app's short name is '#{app['shortname']}' **********")
  Chef::Log.info("********** The app's URL is '#{app['app_source']['url']}'
 **********")
end
```

Of these two approaches, we recommend that you use Chef search. All related examples in this guide demonstrate this approach.

Topics

- App Data Bag (aws_opsworks_app)
- Command Data Bag (aws_opsworks_command)
- Amazon ECS Cluster Data Bag (aws_opsworks_ecs_cluster)
- Elastic Load Balancing Data Bag (aws_opsworks_elastic_load_balancer)
- Instance Data Bag (aws_opsworks_instance)
- Layer Data Bag (aws_opsworks_layer)
- Amazon RDS Data Bag (aws_opsworks_rds_db_instance)
- Stack Data Bag (aws_opsworks_stack)
- User Data Bag (aws_opsworks_user)

App Data Bag (aws_opsworks_app)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

For a Deploy event or an Execute Recipes stack command, represents an app's settings.

The following example shows how to use Chef search to search through a single data bag item and then multiple data bag items to write messages to the Chef log with the apps' short names and source URLs:

```
app = search("aws_opsworks_app").first
Chef::Log.info("********** The app's short name is '#{app['shortname']}' **********")
Chef::Log.info("********** The app's URL is '#{app['app_source']['url']}' **********")

search("aws_opsworks_app").each do |app|
  Chef::Log.info("********** The app's short name is '#{app['shortname']}' **********")
  Chef::Log.info("********** The app's URL is '#{app['app_source']['url']}'
 **********")
```

```
end
```

app_id	app_source	data_sources
deploy	attributes	domains
enable_ssl	environment	name
shortname	ssl_configuration	type

app_id

The app ID (string). A GUID that identifies the app.

app_source

A set of content that specifies the information that OpsWorks Stacks uses to deploy the app from its source control repository. The content varies depending on the repository type.

password

The password for private repositories, and `"null"` for public repositories (string). For private S3 buckets, this content is set to the secret key.

revision

If the repository has multiple branches, the content specifies the app's branch or version, such as `"version1"` (string). Otherwise, it is set to `"null"`.

ssh_key

A deploy SSH key for accessing private Git repositories, and `"null"` for public repositories (string).

type

The app's source location (string). Valid values include:

- `"archive"`
- `"git"`
- `"other"`
- `"s3"`

url

Where the app source is located (string).

user

The user name for private repositories, and `"null"` for public repositories (string). For private S3 buckets, the content is set to the access key.

attributes

A set of content that describes the directory structure and content of the app.

document_root

The root directory of the document tree. Defines the path to the document root–or the location of the app's home page, such as `home_html`–that is relative to your deployment directory. Unless this attribute is specified, the document_root defaults to `public`. The value of `document_root` can start only with `a-z`, `A-Z`, `0-9`, `_` (underscore) or `-` (hyphen) characters.

data_sources

The information required to connect to the app's database. If the app has an attached database layer, OpsWorks Stacks automatically assigns the appropriate values to this content.

The value of data_sources is an array, and arrays are accessed by an integral offset, not by key. For example, to access the app's first data source, use `app[:data_sources][0][:type]`.

database_name

The database name, which is typically the app's short name (string).

type

The database instance's type, typically `"RdsDbInstance"` (string).

arn

The database instance's Amazon Resource Name (ARN) (string).

deploy

Whether the app should be deployed (Boolean). `true` for apps that should be deployed in a Deploy lifecycle event. In a Setup lifecycle event, this content will be `true` for all apps.

To determine which apps should be deployed on an instance, check the layers to which the instance belongs.

domains

A list of the app's domains (list of strings).

enable_ssl

Whether SSL support is enabled (Boolean).

environment

A collection of user-specified environment variables that have been defined for the app. For more information about how to define an app's environment variables, see Adding Apps. Each content name is set to an environment variable name and the corresponding value is set to the variable's value.

name

The app's name, which is used for display purposes (string).

shortname

The app's short name, which is generated by OpsWorks Stacks from the name (string). The shortname is used internally and by recipes; it is used as the name of the directory where your app files are installed.

ssl_configuration

certificate

If you enabled SSL support, the app's SSL certificate; otherwise, `"null"` (string).

chain

If SSL is enabled, content for specifying an intermediate certificate authority key or client authentication (string).

private_key

If you enabled SSL support, the app's SSL private key; otherwise, `"null"` (string).

type

The app's type, which is always set to `"other"` for Chef 12 Linux and Chef 12.2 Windows stacks (string).

Command Data Bag (aws_opsworks_command)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Represents settings for a command that OpsWorks Stacks runs on one or more instances.

The following example shows how to use Chef search to search through a single data bag item and then multiple data bag items to write messages to the Chef log with the commands' types and when they were sent:

```
command = search("aws_opsworks_command").first
Chef::Log.info("********** The command's type is '#{command['type']}' **********")
Chef::Log.info("********** The command was sent at '#{command['sent_at']}' **********")

search("aws_opsworks_command").each do |command|
  Chef::Log.info("********** The command's type is '#{command['type']}' **********")
  Chef::Log.info("********** The command was sent at '#{command['sent_at']}'
 **********")
end
```

args	command_id	iam_user_arn
instance_id	sent_at	type

args

Arguments for the command (string).

command_id

The command's random unique identifier, assigned by OpsWorks Stacks (string).

iam_user_arn

If the command is created by the customer, the Amazon Resource Name (ARN) of the user who created the command (string).

instance_id

The identifier of the instance that the command was run on (string).

sent_at

The timestamp of when OpsWorks Stacks ran the command (string).

type

The command's type (string). Valid values include:

- `"configure"`
- `"deploy"`
- `"deregister"`
- `"execute_recipes"`
- `"grant_remote_access"`
- `"install_dependencies"`
- `"restart"`
- `"revoke_remote_access"`
- `"rollback"`
- `"setup"`
- `"shutdown"`
- `"start"`
- `"stop"`
- `"sync_remote_users"`
- `"undeploy"`
- `"update_agent"`
- `"update_custom_cookbooks"`
- `"update_dependencies"`

Amazon ECS Cluster Data Bag (aws_opsworks_ecs_cluster)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Represents an Amazon ECS cluster's settings.

The following example shows how to use Chef search to search through a single data bag item and then multiple data bag items to write messages to the Chef log with the Amazon ECS clusters' names and Amazon Resource Names (ARNs):

```
ecs_cluster = search("aws_opsworks_ecs_cluster").first
Chef::Log.info("********** The ECS cluster's name is
 '#{ecs_cluster['ecs_cluster_name']}' **********")
Chef::Log.info("********** The ECS cluster's ARN is '#{ecs_cluster['ecs_cluster_arn']}'
 **********")

search("aws_opsworks_ecs_cluster").each do |ecs_cluster|
  Chef::Log.info("********** The ECS cluster's name is
 '#{ecs_cluster['ecs_cluster_name']}' **********")
  Chef::Log.info("********** The ECS cluster's ARN is
 '#{ecs_cluster['ecs_cluster_arn']}' **********")
end
```

ecs_cluster_arn	ecs_cluster_name

ecs_cluster_arn

The cluster's Amazon Resource Name (ARN) (string).

ecs_cluster_name

The cluster's name (string).

Elastic Load Balancing Data Bag (aws_opsworks_elastic_load_balancer)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Represents an Elastic Load Balancing load balancer's settings.

The following example shows how to use Chef search to search through a single data bag item and then multiple data bag items to write messages to the Chef log with the Elastic Load Balancing load balancers' names and DNS names:

```
elastic_load_balancer = search("aws_opsworks_elastic_load_balancer").first
Chef::Log.info("********** The ELB's name is
 '#{elastic_load_balancer['elastic_load_balancer_name']}' **********")
Chef::Log.info("********** The ELB's DNS name is '#{elastic_load_balancer['dns_name']}'
 **********")

search("aws_opsworks_elastic_load_balancer").each do |elastic_load_balancer|
  Chef::Log.info("********** The ELB's name is
 '#{elastic_load_balancer['elastic_load_balancer_name']}' **********")
  Chef::Log.info("********** The ELB's DNS name is
 '#{elastic_load_balancer['dns_name']}' **********")
end
```

elastic_load_balancer_name	dns_name	layer_id

elastic_load_balancer_name

The load balancer's name (string).

dns_name

The load balancer's DNS name (string).

layer_id

The OpsWorks Stacks ID of the layer that the load balancer is assigned to (string).

Instance Data Bag (aws_opsworks_instance)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Represents an instance's settings.

The following example shows how to use Chef search to search through a single data bag item and then multiple data bag items to write messages to the Chef log with the instances' hostnames and IDs:

```
instance = search("aws_opsworks_instance").first
Chef::Log.info("********** The instance's hostname is '#{instance['hostname']}'
 **********")
Chef::Log.info("********** The instance's ID is '#{instance['instance_id']}'
 **********")

search("aws_opsworks_instance").each do |instance|
  Chef::Log.info("********** The instance's hostname is '#{instance['hostname']}'
 **********")
  Chef::Log.info("********** The instance's ID is '#{instance['instance_id']}'
 **********")
end
```

The following example shows different ways of using Chef search to search through multiple data bag items to find the data bag item that contains the specified Amazon EC2 instance ID. The example then uses the data bag item's contents to write a message to the Chef log with the corresponding instance's public IP address:

```
instance = search("aws_opsworks_instance", "ec2_instance_id:i-12345678").first
```

```
Chef::Log.info("********** For instance '#{instance['ec2_instance_id']}', the
 instance's public IP address is '#{instance['public_ip']}' **********")

search("aws_opsworks_instance").each do |instance|
  if instance['ec2_instance_id'] == 'i-12345678'
    Chef::Log.info("********** For instance '#{instance['ec2_instance_id']}', the
 instance's public IP address is '#{instance['public_ip']}' **********")
  end
end
```

The following example shows how to use Chef search with `self:true` to find the data bag item that contains information related to the instance that the recipe is being executed on. The example then uses the data bag item's contents to write a message to the Chef log with the corresponding instance's OpsWorks Stacks-generated ID and the instance's public IP address:

```
instance = search("aws_opsworks_instance", "self:true").first
Chef::Log.info("********** For instance '#{instance['instance_id']}', the instance's
 public IP address is '#{instance['public_ip']}' **********")
```

ami_id	architecture	auto_scaling_type
availability_zone	created_at	ebs_optimized
ec2_instance_id	elastic_ip	hostname
instance_id	instance_type	layer_ids
os	private_dns	private_ip
public_dns	public_ip	root_device_type
root_device_volume_id	self	ssh_host_dsa_key_fingerprint
ssh_host_dsa_key_private	ssh_host_dsa_key_public	ssh_host_rsa_key_fingerprint
ssh_host_rsa_key_private	ssh_host_rsa_key_public	status
subnet_id	virtualization_type	

ami_id

The instance's AMI (Amazon Machine Image) ID (string).

architecture

The instance's architecture, which is always set to `"x86_64"` (string).

auto_scaling_type

The instance's scaling type: `null`, `timer`, or `load` (string).

availability_zone

The instance's Availability Zone (AZ), such as `"us-west-2a"` (string).

created_at

The time that the instance was created, using the UTC "*yyyy*-*mm*-*ddd*T*hh*:*mm*:*ss*+*hh*:*mm*" format (string). For example, `"2013-10-01T08:35:22+00:00"` corresponds to 8:35:22 on Oct. 10, 2013, with no time zone offset. For more information, see ISO 8601.

ebs_optimized

Whether the instance is EBS-optimized (Boolean).

ec2_instance_id

The EC2 instance ID (string).

elastic_ip

The Elastic IP address; set to `"null"` if the instance does not have an Elastic IP address (string).

hostname

The host name, such as `"demo1"` (string).

instance_id

The instance ID, which is an OpsWorks Stacks-generated GUID that uniquely identifies the instance (string).

instance_type

The instance type, such as `"c1.medium"` (string).

layer_ids

A list of the instance's layers, identified by their unique IDs; for example, `307ut64c-c7e4-40cc-52f0-67d5k1f9992c`.

os

The instance's operating system (string). Valid values include:

- "Amazon Linux 2"
- "Amazon Linux 2018.03"
- "Amazon Linux 2017.09"
- "Amazon Linux 2017.03"
- "Amazon Linux 2016.09"
- "Custom"
- "Microsoft Windows Server 2022 Base"
- "Microsoft Windows Server 2022 with SQL Server Express"
- "Microsoft Windows Server 2022 with SQL Server Standard"
- "Microsoft Windows Server 2022 with SQL Server Web"
- "Microsoft Windows Server 2019 Base"
- "Microsoft Windows Server 2019 with SQL Server Express"
- "Microsoft Windows Server 2019 with SQL Server Standard"
- "Microsoft Windows Server 2019 with SQL Server Web"
- "CentOS 7"
- "Red Hat Enterprise Linux 7"
- "Ubuntu 20.04 LTS"
- "Ubuntu 18.04 LTS"
- "Ubuntu 16.04 LTS"
- "Ubuntu 14.04 LTS"

private_dns

The private DNS name (string).

private_ip

The private IP address (string).

public_dns

The public DNS name (string).

public_ip

The public IP address (string).

root_device_type

The root device type (string). Valid values include:

- `"ebs`
- `"instance-store"`

root_device_volume_id

The root device's volume ID (string).

self

`true` if this data bag item contains information about the instance that the recipe is being executed on; otherwise, `false` (Boolean). This value is available only to recipes, not through the OpsWorks Stacks API.

ssh_host_dsa_key_fingerprint

A shorter sequence of bytes that identifies the longer DSA public key (string).

ssh_host_dsa_key_private

The DSA-generated private key for SSH authentication with the instance (string).

ssh_host_dsa_key_public

The DSA-generated public key for SSH authentication with the instance (string).

ssh_host_rsa_key_fingerprint

A shorter sequence of bytes that identifies the longer RSA public key (string).

ssh_host_rsa_key_private

The RSA-generated private key for SSH authentication with the instance (string).

ssh_host_rsa_key_public

The RSA-generated public key for SSH authentication with the instance (string).

status

The instance's status (string). Valid values include:

- `"requested"`
- `"booting"`

- "running_setup"
- "online"
- "setup_failed"
- "start_failed"
- "terminating"
- "terminated"
- "stopped"
- "connection_lost"

subnet_id

The instance's subnet ID (string).

virtualization_type

The instance's virtualization type (string).

Layer Data Bag (aws_opsworks_layer)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Represents a layer's settings.

The following example shows how to use Chef search to search through a single data bag item and then multiple data bag items to write messages to the Chef log with the layers' names and short names:

```
layer = search("aws_opsworks_layer").first
Chef::Log.info("********** The layer's name is '#{layer['name']}' **********")
Chef::Log.info("********** The layer's shortname is '#{layer['shortname']}'
 **********")
```

```
search("aws_opsworks_layer").each do |layer|
  Chef::Log.info("********** The layer's name is '#{layer['name']}' **********")
  Chef::Log.info("********** The layer's shortname is '#{layer['shortname']}'
 **********")
end
```

ecs_cluster_arn	layer_id	name
packages	shortname	type
volume_configurations		

ecs_cluster_arn

If the layer has an Amazon ECS cluster assigned, the Amazon ECS cluster's Amazon Resource Name (ARN) (string).

encrypted

`true` if the EBS volume is encrypted; otherwise, `false` (Boolean).

layer_id

The layer ID, which is a GUID that is generated by OpsWorks Stacks and that uniquely identifies the layer (string).

name

The layer's name, which is used to represent the layer in the console (string). It can be user defined and need not be unique.

packages

A list of packages to be installed (list of strings).

shortname

The layer's shortname, which is user defined (string).

type

The layer's type, which is always set to `"custom"` for Chef 12 Linux and Chef 12.2 Windows (string).

volume_configurations

A list of Amazon EBS volume configurations.

iops

The number of I/O operations per second that the volume can support.

mount_point

The volume's mount point directory.

number_of_disks

The number of disks in the volume.

raid_level

The volume's RAID configuration level.

size

The volume's size in GiB.

volume_type

The volume's type: general purpose, magnetic, provisioned IOPS, throughput-optimized HDD, or cold HDD.

Amazon RDS Data Bag (aws_opsworks_rds_db_instance)

Important

The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

A set of data bag content that specifies an Amazon Relational Database Service (Amazon RDS) instance's configuration as follows:

address	db_instance_identifier	db_password

db_user	engine	rds_db_instance_arn
region		

The following example shows how to use Chef search to search through a single data bag item and then multiple data bag items to write messages to the Chef log with the Amazon RDS instances' addresses and database engine types:

```
rds_db_instance = search("aws_opsworks_rds_db_instance").first
Chef::Log.info("********** The RDS instance's address is
 '#{rds_db_instance['address']}' **********")
Chef::Log.info("********** The RDS instance's database engine type is
 '#{rds_db_instance['engine']}' **********")

search("aws_opsworks_rds_db_instance").each do |rds_db_instance|
  Chef::Log.info("********** The RDS instance's address is
 '#{rds_db_instance['address']}' **********")
  Chef::Log.info("********** The RDS instance's database engine type is
 '#{rds_db_instance['engine']}' **********")
end
```

address

The instance's DNS name.

port

The instance's port.

db_instance_identifier

The instance's ID.

db_password

The instance's master password.

db_user

The instance's master user name.

engine

The instance's database engine, such as `mysql`.

rds_db_instance_arn

The instance's Amazon Resource Name (ARN).

region

The instance's AWS region, such as `us-west-2`.

Stack Data Bag (aws_opsworks_stack)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Represents a stack's settings.

The following example shows how to use Chef search to write messages to the Chef log with the stack's name and cookbook's source url:

```
stack = search("aws_opsworks_stack").first
Chef::Log.info("********** The stack's name is '#{stack['name']}' **********")
Chef::Log.info("********** The stack's cookbook URL is
 '#{stack['custom_cookbooks_source']['url']}' **********")
```

arn	custom_cookbooks_source	name
region	stack_id	use_custom_cookbooks
vpc_id		

arn

The stack's Amazon Resource Name (ARN) (string).

custom_cookbooks_source

A set of content that specify the custom cookbook's source repository.

type

The repository type (string). Valid values include:

- `"archive"`
- `"git"`
- `"s3"`

url

The repository URL, such as `"git://github.com/amazonwebservices/opsworks-demo-php-simple-app.git"` (string).

username

The user name for private repositories, and `null` for public repositories (string). For private Amazon Simple Storage Service (Amazon S3) buckets, the content is set to the access key.

password

The password for private repositories, and `null` for public repositories (string). For private S3 buckets, this content is set to the secret key.

ssh_key

A deploy SSH key for accessing private Git repositories, and `null` for public repositories (string).

revision

If the repository has multiple branches, the content specifies the app's branch or version, such as `"version1"` (string). Otherwise, it is set to `null`.

name

The stack's name (string).

region

The stack's AWS region (string).

stack_id

A GUID that identifies the stack (string).

use_custom_cookbooks

Whether custom cookbooks are enabled (Boolean).

vpc_id

If the stack is running in a VPC, the VPC ID, if the stack is running in a VPC (string).

User Data Bag (aws_opsworks_user)

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Represents a user's settings.

The following example shows how to use Chef search to search through a single data bag item and then multiple data bag items to write messages to the Chef log with the users' user names and Amazon Resource Names (ARNs):

```
user = search("aws_opsworks_user").first
Chef::Log.info("********** The user's user name is '#{user['username']}' **********")
Chef::Log.info("********** The user's user ARN is '#{user['iam_user_arn']}'
 **********")

# Or...

search("aws_opsworks_user").each do |user|
  Chef::Log.info("********** The user's user name is '#{user['username']}' **********")
  Chef::Log.info("********** The user's user ARN is '#{user['iam_user_arn']}'
 **********")
end
```

administrator_privileges	iam_user_arn	remote_access

ssh_public_key	unix_user_id	username

administrator_privileges

Whether the user has administrator privileges (Boolean).

iam_user_arn

The user's Amazon Resource Name (ARN) (string).

remote_access

Whether the user can use RDP to log in to the instance (Boolean).

ssh_public_key

The user's public key, as provided through the OpsWorks Stacks console or API (string).

unix_user_id

The user's Unix ID (number).

username

The user name (string).

OpsWorks Agent Changes

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

Chef 12 Agent Releases

The following table describes important changes to the Chef 12 agent that AWS OpsWorks Stacks installs on instances that it manages.

Agent Version	Description	Release Date
4042	• This agent release only contains minor changes with no new features	February 7, 2023
4041	• This agent release only contains minor changes with no new features • Update Amazon CA certificates	January 27, 2023
4040	• This agent release only contains minor changes with no new features	July 22, 2022
4039	• Fix ECS integration for Ubuntu AMIs	April 30, 2020
4038	• Fix bug when sending instance statistics during DST change • Respect `no_proxy` environment variable during agent download and installation	March 5, 2020
4037	• Add support for signing requests to S3 URLs without region using SigV4 • Remove support for signing S3 requests using SigV2	June 4, 2019
4035	• Fix bug during ECS setup • Fix duplicated `fstab` entries after an instance type change	May 8, 2019
4033	• Add support for Ubuntu 18.04 • Fix agent installation bug in Amazon Linux 2	November 26, 2018
4032	• Added support for Amazon Linux 2	October 24, 2018
4031	• Add support for Amazon Linux 2018.03 • Support public S3 archives hosted on another account	August 15, 2018
4030	• Fix volume handling for c5d instances	May 31, 2018

Agent Version	Description	Release Date
4029	• Install `nvme-cli` on Ubuntu 14.04 • Fix volume mounting on c5, m5 instances • Always preserve hostname on restart	May 2, 2018
4028	• Fix `monit` configuration for CentOS	March 20, 2018
4027	• Support for mounting NVMe volumes on Ubuntu 14.04 (`nvme-cli` must be installed manually) • Do not require `name` property for volumes	February 17, 2018
4026	• Mount NVMe-based EBS volumes using EBS volume ID • Fix EBS volume mounting on i3 instances • Fix order of mounted EBS volumes on c5, m5 instances	January 31, 2018
4025	• Fix handling of NVMe devices	December 13, 2017
4024	• Add support for Amazon Linux 2017.09	December 5, 2017
4023	• Add support for CloudWatch Logs integration	April 2, 2017
4022	• Update Chef client version to 12.18.31	February 1, 2017
4021	• Improve proxy handling	December 16, 2016
4020	• Update Chef client version to 12.16.42	December 8, 2016

Agent Version	Description	Release Date
4019	• Source proxy variables during agent installation • Red Hat Enterprise Linux 7 now uses `systemd` instead of `monit` • Don't setup EPEL on Red Hat Enterprise Linux 7 • Use `flock` instead of `lockrun.c` for process locking • Avoid odd output of `ps -p1` when checking for `systemd`	October 19, 2016
4018	• Update Chef client version to 12.13.37 • Add support for Amazon Linux 2016.09	August 25, 2016
4017	• Update Chef client version to 12.12.15	August 10, 2016
4016	• Fix agent uninstallation on systems where `monit` is not used	June 23, 2016
4015	• Fix ECS setup for Amazon Linux 2016.03	June 17, 2016
4011	• Update Chef client version to 12.10.24 • Improve log upload handling	May 19, 2016
4008	• Add support for Amazon Linux 2016.03 • Add timeout to bundle install • Add xfs to /etc/filesystems if it exists	March 16, 2016
4007	• Update Chef client version to 12.7.2 • Improvements for error handling for on-premises instances (servers hosted outside of AWS) • Improve compatibility with latest chef-sugar • Retry archive download for deployment	March 4, 2016

Agent Version	Description	Release Date
4006	• Update Chef client version to 12.6.0 • Don't install libxml2-devel/libxml2-dev and libxslt-devel/libxslt-dev packages on agent install	January 21, 2016
4005	• Fix ec2 import by always enabling ec2 data in ohai for ec2 infrastructure	December 17, 2015
4004	• OpsWorks Stacks support for Chef 12 Linux- Chef Client 12.5.1	December 3, 2015

Chef 11.10 Agent Releases

The following table describes important changes to the Chef 11.10 agent that AWS OpsWorks Stacks installs on instances that it manages.

Agent Version	Description	Release Date
3456	• This agent release only contains minor changes with no new features • Update Amazon CA certificates	January 27, 2023
3455	• This agent release only contains minor changes with no new features	November 1, 2022
3454	• Fix ECS integration for Ubuntu AMIs	April 28, 2020
3453	• Fix bug when sending instance statistics during DST change • Fix missing packages bug in RHEL7 setup • Respect `no_proxy` environment variable during agent download and installation	March 5, 2020
3452	• Do not include region in the Amazon S3 virtual path URL if it is `us-east-1`	August 13, 2019

Agent Version	Description	Release Date
	• Extract and upload internal cookbooks to stage-region specific buckets • Fix `fstab` entries for Chef 11.10 • Remove SigV2 usage for S3 and get the region for the bucket in the request	
3451	• Add support for Ruby 2.6.1	March 20, 2019
3450	• Fix default EBS attributes • Fix CloudWatchLogs agent installation for Amazon Linux 2 • Fix bundler installation for rubygem versions newer than 2.6.14 • Fix public S3 archives support	December 3, 2018
3449	• Fix volume handling for c5d instances • Fix RAID array support on NVMe-device instances	June 5, 2018
3448	• Upgrade the default 2.3 version of Ruby to 2.3.7 • Fix mounting EBS volumes on NVMe based instances on Ubuntu 14.04 instances • Support public Amazon S3 archives hosted on another account • Fix `opsworks-agent` boot issues on Red Hat Enterprise Linux instances	May 8, 2018
3447	• Mount NVMe-based EBS volumes using EBS volume ID • Fix EBS volume mounting on i3 instances • Fix order of mounted EBS volumes on c5, m5 • Update the default 2.3 version of Ruby to 2.3.6	January 31, 2018
3446	• Fix handling of NVMe devices • Update the default 2.3 version of Ruby to 2.3.5	December 14, 2017

Agent Version	Description	Release Date
3445	• Add support for Amazon Linux 2017.09 • Update the default 2.2 version of Ruby to 2.2.8	October 31, 2017
3444	• Add support for CloudWatch Logs	April 1, 2017
3443	• Improve proxy handling	December 15, 2016
3442	• Update the default 2.3 version of Ruby to 2.3.3 • Update the default 2.2 version of Ruby to 2.2.6	December 6, 2016
3441	• Source proxy variables during agent installation	October 21, 2016
3440	• Add support for Amazon Linux 2016.09	September 13, 2016
3439	• Minor changes; no new features	July 29, 2016
3438	• Add support for Ruby 2.3.1 • Improve instance registration with credentials from IAM instance profile • Remove `s3curl.pl` leftovers • Fix ECS setup for Amazon Linux 2016.03	June 17, 2016
3437	• Update the default 2.2 version of Ruby to 2.2.5	May 4, 2016
3436	• Update EPEL URL for Red Hat Enterprise Linux. IMPORTANT: without this change, Red Hat Enterprise Linux instances fail to boot.	April 18, 2016
3435	• Update the default 2.1 version of Ruby to 2.1.9 • Improve handling of Amazon S3 and archive deployments	April 6, 2016
3434	• Add support for Amazon Linux 2016.03 • Retry package installations	March 16, 2016

Agent Version	Description	Release Date
3433	• Some improvements for on-premises instances (servers hosted outside of AWS) • Improve compatibility with latest `chef-sugar` • Retry archive download for deployment • Fix Ruby gems installation URL	February 27, 2016
3432	• Improve handling of special characters in bucket names • Update `s3_file` to version 2.6.6 • Skip mounting of volumes with no specified mount point • Always restart `unicorn` instead of a stop and start to prevent downtime during deploys • Always update custom cookbook for `setup` command • After creating RAID arrays update `initramfs` to prevent device mapping issues on reboot	January 20, 2016
3431	• Fixed `passenger` and `unicorn` gem installation issue in Rails layer • Updating the default 2.0, 2.1 and 2.2 versions of Ruby to 2.0.0p648, 2.1.8 and 2.2.4 • Allow `postgres` package names to set in custom JSON • Update the Node.js default version to 0.12.9	December 22, 2015
3430	• Minor changes; no new features	November 25, 2015
3429	• Improve OpsWorks agent daemonize (close stdout/stderr) • Improve robustness of `s3_file` resource (retries, caught exceptions)	November 18, 2015

Agent Version	Description	Release Date
3428	• Adding `postgres` adapter detection based on the Gemfile, fixes https://github.com/aws/opsworks-cookbooks/issues/136	June 17, 2016
3427	• Fixed an issue with retrieving credentials in the agent • Updating the default 2.0, 2.1 and 2.2 versions of Ruby to 2.0.0p647, 2.1.7 and 2.2.3	September 11, 2015
3426	• Updated `aws-sdk` to 1.65.0 • Improving download from Amazon S3 by replacing `s3curl` with `s3_file cookbook` • Change the default Node.js version to 0.12.7 • Logging added for Node.js apps. STDOUT and STDERR logged and rotated in the `shared/log` directory • Make custom cookbook submodule checkout update explicit • Added workaround for https://github.com/aws/opsworks-cookbooks/issues/213 that will check to ensure bind mounts have been made before the `deploy` directory is created	August 27, 2015
3425	• ECS support for Amazon Linux and Ubuntu	July 27, 2015
3424	• Minor changes; no new features	July 9, 2015
3422	• Full support of Red Hat Enterprise Linux 7 • Make `/etc/hosts` generation more resilient to errors	June 29, 2015

Agent Version	Description	Release Date
3421	• Option to override database package name for Red Hat Enterprise Linux 7 • Updated the `monit systemd` config to prevent `systemd` from sending the `kill` signal to processes monitored by `monit`	June 11, 2015

OpsWorks Stacks Resources

> **Important**
>
> The AWS OpsWorks Stacks service reached end of life on May 26, 2024 and has been disabled for both new and existing customers. We strongly recommend customers migrate their workloads to other solutions as soon as possible. If you have questions about migration, reach out to the AWS Support Team on AWS re:Post or through AWS Premium Support.

The following related resources can help you as you work with this service.

Reference Guides, Tools, and Support Resources

Several helpful guides, forums, contact info, and other resources are available from OpsWorks Stacks and Amazon Web Services.

- **OpsWorks Stacks API Reference** – Descriptions, syntax, and usage examples about OpsWorks Stacks actions and data types, including common parameters and error codes.
- **OpsWorks Stacks Technical FAQ** – Top questions developers have asked about this product.
- **OpsWorks Stacks Release Notes** – A high-level overview of the current release. This document specifically notes any new features, corrections, and known issues.
- **AWS Tools for PowerShell** – A set of Windows PowerShell cmdlets that expose the functionality of the AWS SDK for .NET in the PowerShell environment.
- **AWS Command Line Interface**– A uniform command line syntax for accessing AWS services. The AWS CLI uses a single setup process to enable access for all supported services.
- **OpsWorks Stacks Command Line Reference** – OpsWorks Stacks-specific commands for use at a command line prompt.

- Classes & Workshops – Links to role-based and specialty courses, in addition to self-paced labs to help sharpen your AWS skills and gain practical experience.
- AWS Developer Center – Explore tutorials, download tools, and learn about AWS developer events.

- AWS Developer Tools – Links to developer tools, SDKs, IDE toolkits, and command line tools for developing and managing AWS applications.
- Getting Started Resource Center – Learn how to set up your AWS account, join the AWS community, and launch your first application.
- Hands-On Tutorials – Follow step-by-step tutorials to launch your first application on AWS.
- AWS Whitepapers – Links to a comprehensive list of technical AWS whitepapers, covering topics such as architecture, security, and economics and authored by AWS Solutions Architects or other technical experts.
- AWS Support Center – The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.
- Support – The primary webpage for information about Support, a one-on-one, fast-response support channel to help you build and run applications in the cloud.
- Contact Us – A central contact point for inquiries concerning AWS billing, account, events, abuse, and other issues.
- AWS Site Terms – Detailed information about our copyright and trademark; your account, license, and site access; and other topics.

AWS Software Development Kits

Amazon Web Services provides software development kits for accessing OpsWorks Stacks from several different programming languages. The SDK libraries automate a number of common tasks, including cryptographically signing your service requests, retrying requests, or handling error responses.

- **AWS SDK for Java** – Setup and other documentation
- **AWS SDK for .NET** – Setup and other documentation.
- **AWS SDK for PHP** – Documentation
- **AWS SDK for Ruby** – Documentation
- other documentation
- **AWS SDK for Python (Boto)** – Setup and other documentation

Open Source Software

OpsWorks Stacks includes a variety of open-source software packages, which are governed by their respective licenses. For more information, see the following:

- For Chef 12 Linux instances, open the `THIRD_PARTY_LICENSES` file in the `/opt/aws/opsworks/current` directory on the instance.
- For Chef 11.10 and earlier versions for Linux, download the OpsWorks Linux Agent Attributions Document PDF.

OpsWorks document history

Change	Description	Date
Deprecated AWS Managed Policies	The following AWS Managed Policies are deprecated because AWS OpsWorks for Chef Automate is deprecated: `AWSOpsWorksCMServiceRole` and `AWSOpsWorksCMInstanceProfileRole`.	May 26, 2025
Updates to AWS OpsWorks Stacks	You can now use the Detach in Place tool to detach your OpsWorks instances from the OpsWorks Stacks service, see Using the AWS OpsWorks Stacks Detach in Place tool in this guide.	April 11, 2024
Updates to AWS OpsWorks Stacks	You can now migrate your AWS OpsWorks Stacks to AWS Systems Manager Application Manager by using a migration script. For more information, see Migrating your AWS OpsWorks Stacks applications to AWS Systems Manager Application Manager in this guide.	December 22, 2022
Updates to AWS OpsWorks for Chef Automate and AWS OpsWorks for Puppet Enterprise	A troubleshooting procedure is now available that describes what you can do if system maintenance fails	September 29, 2022

	for your AWS OpsWorks for Chef Automate or OpsWorks for Puppet Enterprise server. For more information, see System maintenance fails for Chef Automate server or System maintenance fails for Puppet Enterprise server. in this guide.	
Updates to AWS OpsWorks for Chef Automate and AWS OpsWorks for Puppet Enterprise	A troubleshooting procedure is now available if your AWS OpsWorks for Chef Automate or OpsWorks for Puppet Enterprise server enters a `Connection lost` state. For more information, see Chef Automate server is in a `Connection lost` state or Puppet Enterprise server is in a `Connection lost` state in this guide.	March 23, 2022
Updates to AWS OpsWorks Stacks	As a security best practice, you can now add an `aws:SourceArn` or `aws:SourceAccount` condition key (or both) to trust relationship policies that allow AWS OpsWorks Stacks access to perform tasks in other AWS services. For more information, see Cross-service confused deputy prevention in AWS OpsWorks Stacks in this guide.	March 4, 2022

Updates to AWS OpsWorks for Chef Automate and AWS OpsWorks for Puppet Enterprise	As a security best practice, you can now add an `aws:SourceArn` or `aws:SourceAccount` condition key (or both) to trust relationship policies that allow AWS OpsWorks for Chef Automate and OpsWorks for Puppet Enterprise access to perform tasks in other AWS services. For more informati on, see Cross-service confused deputy prevention in this guide.	January 10, 2022
Updates to AWS OpsWorks for Chef Automate and AWS OpsWorks for Puppet Enterprise	AWS OpsWorks for Chef Automate and OpsWorks for Puppet Enterprise have updated the managed policies `AWSOpsWorksCMServi ceRole` and `AWSOpsWor ksCMInstanceProfil eRole`, and now store secrets in AWS Secrets Manager.	May 3, 2021

Updates to AWS OpsWorks for Puppet Enterprise	The engine version of an OpsWorks for Puppet Enterprise server that you create in the console is now 2019.8.5. By using the API, you can specify either version `2019` or `2017` when you create a Puppet Enterpris e server. The `DescribeS ervers` API now returns an attribute called `PUPPET_AP I_CRL` in its results. This attribute contains a certifica te revocation list for internal use.	April 28, 2021
OpsWorks Stacks uses a new managed policy	OpsWorks Stacks has changed the managed policy that includes permissio ns to perform all actions in OpsWorks Stacks. The new policy is **AWSOpsWor ks_FullAccess**. For more information about the permissions in this policy, see Example policies.	February 19, 2021
Migrate AWS OpsWorks Stacks stacks from EC2-Class ic to a VPC	Documentation has been added describing how to migrate an AWS OpsWorks Stacks stack from EC2-Classic to a VPC.	September 29, 2020

Regenerate a starter kit for AWS OpsWorks for Chef Automate and AWS OpsWorks for Puppet Enterprise	Documentation has been added describing how to regenerate the starter kit for an AWS OpsWorks for Chef Automate or an AWS OpsWorks for Puppet Enterprise server.	July 29, 2020
AWS OpsWorks for Puppet Enterprise lets you create a server that uses a custom domain, certificate, and private key	You can now create an OpsWorks for Puppet Enterprise server that uses a custom domain, certificate, and private key. You can update an existing Puppet Enterprise server to use a custom domain by creating a server from a backup of an existing server.	April 17, 2020
AWS OpsWorks for Chef Automate and AWS OpsWorks for Puppet Enterprise now support tagging in the console	You can now add tags to an AWS OpsWorks for Chef Automate server or an AWS OpsWorks for Puppet Enterprise master, or to server backups, by using either the AWS Management Console or the AWS CLI. For more information, see Work with Tags (Chef) or Work with Tags (Puppet).	February 26, 2020

AWS OpsWorks for Chef Automate simplifies upgrade of existing Chef Automate 1 servers to Chef Automate 2	You can upgrade eligible AWS OpsWorks for Chef Automate servers running Chef Automate 1 to Chef Automate 2 by choosing **Start upgrade** on your server's details page in the console, or by running the `StartMaintenance` API action. For more informati on, see Upgrade an AWS OpsWorks for Chef Automate Server to Chef Automate 2.	January 24, 2020
AWS OpsWorks for Chef Automate and AWS OpsWorks for Puppet Enterprise	A new chapter about Security in OpsWorks CM (AWS OpsWorks for Chef Automate and AWS OpsWorks for Puppet Enterprise) has been added to the guide.	December 23, 2019
AWS OpsWorks for Chef Automate and AWS OpsWorks for Puppet Enterprise support tagging	You can now add tags to an AWS OpsWorks for Chef Automate server or an AWS OpsWorks for Puppet Enterprise master, or to server backups, by using the AWS CLI. OpsWorks CM now supports tag-based authoriza tion.	December 18, 2019

AWS OpsWorks for Chef Automate lets you create a server that uses a custom domain, certificate, and private key	You can now create an AWS OpsWorks for Chef Automate 2.0 server that uses a custom domain, certificate, and private key. You can update an existing Chef Automate 2.0 server to use a custom domain by creating a server from a backup of an existing server.	October 22, 2019
OpsWorks Stacks now supports Ruby 2.6.1	OpsWorks Stacks supports Ruby 2.6.1 on Rails App Server layers in Chef 11.10 stacks.	May 2, 2019
AWS OpsWorks for Chef Automate now supports Chef Automate 2.0	New AWS OpsWorks for Chef Automate servers will run Chef Automate 2.0, which includes updates to Chef InSpec, new features in compliance scanning and reporting, and Chef Infra.	April 30, 2019
AWS OpsWorks for Chef Automate and AWS OpsWorks for Puppet Enterprise	You can now use AWS CloudFormation to create an AWS OpsWorks for Chef Automate server or an AWS OpsWorks for Puppet Enterprise master server.	January 24, 2019
OpsWorks Stacks	OpsWorks Stacks now supports instances running Ubuntu 18.04 LTS in Chef 12 stacks.	December 18, 2018

AWS OpsWorks for Puppet Enterprise	Added procedure for setting up an SSH-based connection to a control repository that uses CodeCommit.	December 3, 2018
OpsWorks Stacks	OpsWorks Stacks now supports instances running Amazon Linux 2 in Chef 12 stacks.	November 15, 2018
OpsWorks Stacks	OpsWorks Stacks now supports instances running Amazon Linux 2018.03 in Chef 11.10 stacks.	October 23, 2018
OpsWorks Stacks	OpsWorks Stacks now supports instances running Amazon Linux 2018.03 in Chef 12 stacks.	August 23, 2018
AWS OpsWorks for Chef Automate and OpsWorks for Puppet Enterprise	OpsWorks for Puppet Enterprise has upgraded to PE 2018.1.2. AWS OpsWorks for Chef Automate has upgraded to Chef Automate 1.8.68.	June 29, 2018

- **AWS OpsWorks for Chef Automate and OpsWorks for Puppet Enterprise API version:** 2016-11-01
- **OpsWorks Stacks API version:** 2016-03-08
- **Latest documentation update:** 2024-04-11

Earlier updates

The following table describes important changes in each release of the *OpsWorks User Guide* before June 2018.

Description	Date
OpsWorks Stacks Chef version for Windows-based stacks upgraded to 12.22; Ruby version is now 2.3.6.	April 19, 2018
New procedures for creating an AWS OpsWorks for Chef Automate server or an OpsWorks for Puppet Enterprise master by using the AWS CLI.	March 23, 2018
Chef Automate version updated to 1.8; Chef Compliance setup simplified with the addition of the `opsworks-audit` cookbook.	March 5, 2018
Added support for OpsWorks Stacks events in Amazon CloudWatch Events.	February 20, 2018
Added support for new EBS volume types in OpsWorks Stacks, and a new API, DescribeOperatingSystems.	January 25, 2018
OpsWorks for Puppet Enterprise and AWS OpsWorks for Chef Automate now support selecting multiple security groups when you are creating a server.	January 18, 2018
Added support for OpsWorks Stacks in the Europe (Paris) Region.	December 19, 2017
Added support for AWS OpsWorks for Chef Automate and OpsWorks for Puppet Enterprise in six additional regions, and added procedure s for creating backups of AWS OpsWorks for Chef Automate and OpsWorks for Puppet Enterprise servers in the AWS Management Console.	December 18, 2017
Added the new OpsWorks for Puppet Enterprise service and documentation.	November 16, 2017
Added support for Amazon Linux 2017.09 to OpsWorks Stacks.	November 7, 2017
Added support for Chef Compliance to AWS OpsWorks for Chef Automate.	October 25, 2017

Description	Date
Added support for Amazon Linux 2017.09 to AWS OpsWorks for Chef Automate.	October 9, 2017
Added System Maintenance topic to AWS OpsWorks for Chef Automate chapter.	July 28, 2017
Added support for tags in OpsWorks Stacks.	June 6, 2017
Added integration with CloudWatch Logs.	April 10, 2017
Added the new AWS OpsWorks for Chef Automate service and documentation.	December 1, 2016
Added support for the US East (Ohio) Region regional endpoint.	October 12, 2016
Added support for stacks and instances that run the Amazon Linux 2016.09 operating system.	September 30, 2016
Added support for the Asia Pacific (Seoul) Region and nine additiona l regional endpoints.	August 15, 2016
Added support for Node.js 0.12.15 and Ruby 2.3 in built-in layers.	July 6, 2016
Added support for the Asia Pacific (Mumbai) Region.	June 28, 2016
Added support for stacks and instances that run the CentOS 7 operating system.	June 22, 2016
Added walkthrough describing CodePipeline and OpsWorks Stacks integration.	June 2, 2016
Added support for stacks and instances that run the Ubuntu 16.04 LTS operating system.	June 1, 2016
Added Chef 12 Linux support and related documentation.	December 3, 2015
Added Node.js walkthrough to Getting Started.	July 14, 2015
Added two new cookbook examples to Cookbooks 101.	July 14, 2015

Description	Date
Added support for agent version management.	June 23, 2015
Added support for managing the agent version.	June 24, 2015
Added support for custom Windows AMIs.	June 22, 2015
Added three new Best Practices topics.	June 11, 2015
Added support for Windows stacks.	May 18, 2015
Added a Best Practices chapter.	Dec. 15, 2014
Added support for Elastic Load Balancing connection draining and custom Shutdown timeouts.	Dec. 15, 2014
Added support for registering instances created outside of AWS OpsWorks Stacks.	Dec 9, 2014
Added support for Amazon SWF.	Sept. 4, 2014
Added support for associating environment variables with apps and extended Cookbooks 101.	July 16, 2014
Added Cookbooks 101, a tutorial introduction to implementing cookbooks.	July 16, 2014
Added support for CloudTrail.	June 4, 2014
Added support for Amazon RDS.	May 14, 2014
Added support for Chef 11.10 and Berkshelf.	Mar. 27, 2014
Added support for Amazon EBS PIOPS volumes.	Dec. 16, 2013
Added resource-based permissions.	Dec. 5, 2013
Added resource management.	Oct. 7, 2013
Added support for VPCs.	Aug. 29, 2013

Description	Date
Added support for custom AMIs and Chef 11.4.	July 24, 2013
Added console support for multiple layers per instance.	July 1, 2013
Added support for Amazon EBS-backed instances, Elastic Load Balancing, and Amazon CloudWatch monitoring.	May 14, 2013
Initial release of the AWS OpsWorks Stacks User Guide.	February 18, 2013